2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A MISSION PLANNING EXPERT SYSTEM
WITH THREE-DIMENSIONAL PATH OPTIMIZATION
FOR THE NPS MODEL 2
AUTONOMOUS UNDERWATER VEHICLE

by

Seow Meng Ong

June, 1990

Thesis Advisor: Se-Hung Kwak

Approved for public release; distribution is unlimited.

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report
2b Declassification/Downgrading Schedule	Approved for public release; distribution is unlimited.
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization Naval Postgraduate School; 6b Office Symbol *(If Applicable)* 52	7a Name of Monitoring Organization Naval Postgraduate School
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol *(If Applicable)*	9 Procurement Instrument Identification Number
8c Address *(city, state, and ZIP code)*	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

11 Title *(Include Security Classification)* A MISSION PLANNING EXPERT SYSTEM WITH THREE-DIMENSIONAL PATH OPTIMIZATION FOR THE NPS MODEL 2 AUTONOMOUS UNDERWATER VEHICLE.

12 Personal Author(s) Seow Meng Ong

13a Type of Report	13b Time Covered	14 Date of Report *(year, month,day)*	15 Page Count
Master's Thesis	From Sep 1989 To June 1990	June 1990	192

16 Supplementary Notation The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	Mission Planning, Mission Control, Path-Planning, Path-Search, Heuristic search, Autonomous Underwater Vehicle, Autnomous Vehicle, Expert System

19 Abstract *(continue on reverse if necessary and identify by block number*

Unmanned vehicle technology has matured significantly over the last two decades. This is evidenced by its widespread use in industrial and military applications ranging from deep-ocean exploration to anti-submarine warfare. Indeed, the feasibility of short range, special-purpose vehicles (whether autonomous or remotely operated) is no longer in question. The research efforts have now begun to shift their focus on development of reliable, longer range, high-endurance and fully autonomous systems. One of the majot underlying technologies required to realize this goal is Artificial Intelligence (AI). The latter offers great potential to endow vehicles with the intelligence needed for full autonomy and extended range capability; this involves the increased application of AI techniques to support mission planning and execution, navigation and contigency planning.

This thesis addresses two issues associated with the above goal for Autonomous Underwater Vehicles (AUV's). Firstly, a new approach is proposed for path planning in underwater environments that is capable of dealing with uncharted obstacles and which requires significantly less planning time and computer memory. Secondly, it explores the use of expert system technology in the planning of AUV missions.

(KR) (

20 Distribution/Availability of Abstract [✓ X unclassified/unlimited] ☐ same as report ☐ DTIC users	21 Abstract Security Classification Unclassified
22a Name of Responsible Individual Prof. Se-Hung Kwak	22b Telephone *(Include Area code)* (408) 646-2168; 22c Office Symbol 52KW

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted security classification of this page

All other editions are obsolete Unclassified

Approved for public release; distribution is unlimited.

A MISSION PLANNING EXPERT SYSTEM WITH THREE-DIMENSIONAL PATH OPTIMIZATION FOR THE NPS MODEL 2 AUTONOMOUS UNDERWATER VEHICLE

by

Seow Meng Ong
B. Eng., National University of Singapore, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1990

Author: [illegible]
Seow Meng Ong

Approved by: Se-Hung Kwak
Se-Hung Kwak, Thesis Advisor

Robert B. McGhee
Robert B. McGhee, Second Reader

Robert B. McGhee
Robert B. McGhee, Chairman, Department of Computer Science

ABSTRACT

Unmanned vehicle technology has matured significantly over the last two decades. This is evidenced by its widespread use in industrial and military applications ranging from deep-ocean exploration to anti-submarine warfare. Indeed, the feasibility of short-range, special-purpose vehicles (whether autonomous or remotely operated) is no longer in question. The research efforts have now begun to shift their focus on development of reliable, longer-range, high-endurance and fully autonomous systems. One of the major underlying technologies required to realize this goal is Artificial Intelligence (AI). The latter offers great potential to endow vehicles with the intelligence needed for full autonomy and extended range capability; this involves the increased application of AI techniques to support mission planning and execution, navigation and contingency planning.

This thesis addresses two issues associated with the above goal for Autonomous Underwater Vehicles (AUV's). Firstly, a new approach is proposed for path planning in underwater environments that is capable of dealing with uncharted obstacles and which requires significantly less planning time and computer memory. Secondly, it explores the use of expert system technology in the planning of AUV missions.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The original code for the NPS AUV graphical simulator to which enhancements were made during this research, was developed by Dave Marco, Ray Rogers, and Mike Schwartz. The network communications software was not modified and was written by Ted Barrow, Se-Hung Kwak, Bill Teter, and Larry Shannon.

I would like to thank Professor Se-Hung Kwak for his invaluable advice and assistance rendered especially during the development work on the LISP machines. I am also grateful to Professor Robert McGhee for his constructive comments and overall guidance to the project. Both professors have been very encouraging and supportive and it has been a real pleasure working with them.

My sincere thanks also to Mr. Albert Wong and his colleagues of the Computer Science Department Technical Support Staff for their professional support in debugging the communications software.

Most of all, I thank my wife and son for their unfailing love, support and understanding, without which the success of this work would not have been possible.

I. INTRODUCTION

A. BACKGROUND

The last two decades witnessed significant progress in unmanned vehicle technology. This, coupled with advances in computer and Artificial Intelligence (AI) research has increased the likelihood of realizing effective unmanned autonomous undersea vehicles in the near future. With the maturity in the basic technologies required, the research focus has begun to shift towards the development of more reliable, longer range, higher endurance and fully autonomous systems. In line with these developments, the Naval Postgraduate School (NPS) is currently constructing an experimental Autonomous Underwater Vehicle (AUV) to support research on the technology issues related to the above challenge.

In conjunction with research efforts on the vehicle design, previous student thesis studies [Ref. 1, 2] have also centered on the creation of a "laboratory testbed environment" for testing AUV mission planning, navigation, and control issues using simulated environments. The testbed is comprised of a visual simulator (a high-resolution graphics workstation) linked to a special-purpose AI workstation. The latter is used for prototyping AI software for mission planning and control while the visual simulator facilitates 3-D visualization of the AUV behavior during tests. The whole setup is aimed at providing effective and quick feedback on the results and thus reducing the overall time and expense of AUV subsystem development.

This thesis is devoted to the investigation of two inter-related issues, namely, mission planning and path planning for Autonomous Underwater Vehicles, both of which are issues central to the development of completely autonomous vehicles. In the process, the laboratory testbed mentioned above is used to demonstrate and validate the results.

B. MISSION PLANNING EXPERT SYSTEM

Mission plans can be constructed at different levels of abstraction. At the highest levels, they are mission specifications, detailing the mission objectives, the mission tasks and the constraints under which the mission is to be executed. At the lower levels, they list the phases of the mission and detail the tactical actions to be taken in each phase. The task of transforming the high-level mission specifications to low-level plans is presently done by the human mission planner. However, with the growing maturity of expert systems technology, it has become increasingly feasible to develop systems that automatically perform this translation.

In AUV applications, a major output of the planning process is the route or path to be taken by the vehicle. The path derived should be consistent with the high-level mission objectives and, in particular, should satisfy the mission constraints. The complexity of this task depends on the number and type of constraints. The latter can be imposed by the vehicle, by the environment in which it is to operate, and by the nature of the mission. Vehicle-related constraints result from the physical characteristics of the vehicle (such as size and weight), its dynamics (and hence maneuverability), and the degree of control available. Environmental constraints can be natural or man-made; for instance, a minefield presents as much an obstacle to a vehicle as rough undersea terrain. Finally, the nature of the mission refers to factors such as the time-urgency of the mission, the need for stealth (detection avoidance) or for threat avoidance.

Fortunately, many path-search algorithms exist in the AI field [Ref. 3, 4, 5, 6, 7, 8], each having its inherent advantages and disadvantages. As will be explained in the next section, some algorithms provide optimal shortest path solutions, while others minimize the time required for planning. However, since it is inappropriate for the human planner to be thoroughly familiar with the characteristic strengths and weaknesses of all available algorithms, the use of an automated planning tool would be highly desirable. This thesis explores one approach to designing an expert system that selects the best path-planning algorithm for the mission, based on the projected balance between mission factors such as time, energy, risk, etc.

C. PATH PLANNING

1. ROUTE PLANNING vs PATH PLANNING

Path planning aims at deriving a well defined path for the vehicle that satisfies the constraints and requirements of a mission. This can be done in two stages, first at the macro-level and then at the micro-level. In order to differentiate between the two, henceforth, the macro-level path planner shall be referred to as the ***route planner***, and the micro-level route planner as the ***path planner***. Macro-level route planning takes a macroscopic view of the area of operation by partitioning it into regions such as sonobouy fields, unnavigable areas, minefields, search areas, and so forth. To do this, a priori intelligence information concerning the environment may be required. The best route, made up of a sequence of joined path-segments passing through or avoiding specific regions and satisfying the high-level *mission objectives* and constraints, is then determined and selected from among possible alternatives.

The requirement for a micro-level path planner is dependent on the *agent* that will ultimately traverse the route. By agent is meant some entity capable of independent motion along a given path. If the agent is man, then the output of the route planner would be sufficient. However, for a land-based autonomous robot or vehicle, for instance, this is inadequate since it must also account for micro-level problems such as avoiding pits, local steep slopes and physical objects along the path. Thus the role of the path-planner is to derive a detailed path for each path-segment of the route chosen. This can be done in the pre-execution phases and then modified as necessary during execution whenever unforseen events or obstacles are encountered.

2. SEARCH METHODS IN PATH PLANNING

Invariably, some form of ***search*** [Ref. 9] technique in the Artificial Intelligence domain is employed in path-planning. Search can be defined as the systematic exploration of the different possibilities that potentially offer a solution. Many search strategies exist, the classical ones being Depth-first, Breadth-first, Best-first, A* [Ref. 9, 10], etc. Variants of these have also been used in numerous applications. In determining the suitability of a search technique, there are two important application-related factors which must be considered - the **size of the search space** and the **availability of a priori information** concerning the environment.

The practicality of a search method is highly dependent on the size of the search space because of the physical limitations in the computational time and space resources of a computer. For instance, exhaustive search techniques, such as the breadth-first strategy, are not practical for applications with a large search space.

One measure of the size of a search space is the ***branching factor*** [Ref. 9, 10], which is defined as the average number of alternatives at each decision point or node (the average number of successors possessed by each node) in a decision tree. For instance, in a 2-dimension path-planning problem, each position on a rectangular grid has 8 neighbors resulting in a branching factor of 8. Heuristics are often used to reduce the branching factor, thereby making feasible an otherwise impractical technique. In the underwater environment, however, the problem is compounded by an additional dimension. Unlike two-dimensional path-finding problems, each location on a three-dimensional grid has 26 alternatives (Figure 5.1). A massive but intelligent pruning of the search tree is therefore required, if a technique is to be viable.

The second factor - the availability of a priori information concerning the environment - partitions search methods for path-planning into two categories:

1. Methods which require a priori terrain/environment information. Most classical search techniques and their variants fall exclusively under this category.

2. Methods which do not require such a priori information. The methods in this category inevitably, require some form of sensing devices, such as vision sensors, ultrasonic sensors or contact sensors. In reality, situations possessing complete a priori information on the environment or terrain are few. Even where a priori information is available, such data may not be accurate or complete due to the dynamic nature of the environment. Examples include enemy territory and uncharted areas. Thus, if vehicles are to be completely autonomous, they must be endowed with the capability to perform without complete information. Published work relating to this area is scarce.

D. SCOPE OF THESIS

This study is focussed specifically on three objectives:

1. To present the software design of a mission planning expert system which transforms high-level mission specifications into detailed low-level plans.
2. To develop a viable path-search strategy for underwater environments, called Heuristic Search.
3. To compare the performance of 3 different search strategies for path planning, namely, Best-first, A*, and Heuristic search.

E. THESIS ORGANIZATION

Since this thesis has two distinct parts, namely, the design of a Mission Planning Expert System, and the design of Heuristic Search strategy for path-planning, this thesis document could either adopt a bottom-up or a top-down approach to describing the work. After much deliberation, it was decided that a top-down approach would be advantageous in helping the reader to better appreciate the low-level details of path planning, if an overview of the system is first presented.

Chapter II reviews previous and ongoing work in the area of mission planning and control for AUV's, and in the area of path planning search methods. In particular, the different system architectures that have been proposed for mission planning and control are briefly described.

Chapter III presents a detailed problem statement for this thesis. First, the physical characteristics of the current vehicle and the proposed control architecture are discussed in order to provide an overview of the system. The models and assumptions on which this thesis is based are then presented together with a description of the laboratory testbed simulator.

Chapter IV presents the internals of the Mission Planning Expert System. It expounds on the top-level software architecture and explains how each entity is represented within the system. It then proceeds with a description of the detailed design for each functional component. Finally, the reader is led through an illustrative example of how a specific mission is planned using the Mission Planning Workstation developed.

Chapter V describes the methodology of the Heuristic path-search strategy. It explains each component concept in detail, and shows how it influences the vehicle's decision on the path to take to reach the goal. Chapter VI follows up with a comparative study of the three path-search strategies, namely A*, Best-first, and Heuristic search. The detailed results of several simulations, which were run in order to derive their relative performances, are presented.

Finally, Chapter VII summarizes the contributions of this thesis and suggests further extensions to the project.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

The ultimate research goal in the area of mission planning and control for AUV's is to enable a vehicle to operate autonomously without human intervention in its fulfillment of a given mission. This can only be achieved if the vehicles are endowed with the intelligence required to respond to, or deal with, unforseen situations. The realization of such behavior involves automating some of the important high-level functions, such as planning, planning-control, and decision-making, which are ordinarily undertaken by a human planner. To satisfy this objective, the ardent efforts of the AUV research community have resulted in a variety of innovative strategies and corresponding system architectures for mission planning and control. The major ones are discussed in this chapter.

In the domain of path planning and navigation for autonomous vehicles in general, as will be seen, much of the early research work relied on several fundamental premises. Firstly, most previous research is targeted for robotic applications in two-dimensional environments. Secondly, all obstacles are typically approximated by polyhedral shapes to simplify the algorithm. A third fundamental assumption is that a priori information on the environment is available; where this is untrue, the algorithms proposed require the robots to first "learn" about the environment, and to form its own model concerning the world [Ref. 11], prior to actual navigation.

B. ARCHITECTURES FOR MISSION PLANNING AND CONTROL

1. BLACKBOARD BASED SYSTEMS

In recent years, there has been considerable interest in the use of "blackboard" architectures as the structural design paradigm for knowledge-based control architectures onboard AUV's [Ref. 12, 13, 14]. The methodology derives its name from the organized global data space where all system data is placed: the blackboard. An example is the ongoing research work at the Marine Systems Engineering Laboratory (MSEL) at the University of New Hampshire [Ref. 12], where a Blackboard Control Architecture (BCA) is used for an experimental AUV route planner, named the "Supervisor". The focus of the work is on route planning. Given a high level mission specification consisting of an unordered list of way-points to visit and surveys to run, the system works out a route connecting the mission tasks and issues intermediate level motion commands that describe the route.

The Supervisor views a route problem as two distinct problems: the domain problem of actually planning a route and the control problem of how to go about planning the route. Thus, a dual blackboard architecture is used to separate the two, resulting in two distinct components in the system: the route planning subsystem and the control planning subsystem. Each component is comprised of a blackboard and a pool of knowledge sources. The knowledge source pools possess the procedural knowledge while the data generated and used by those knowledge sources is "written" on the blackboards. A knowledge source is an independent process that acts as a specialist in some particular area of the problem. Knowledge sources have a condition/action format. They "trigger" and become executable if their conditions evaluate to true, in which case, a Knowledge Source Activation Record (KSAR) is generated and stored in an agenda of KSARs waiting to be executed. When selected for running, the action portion is executed and any output from it is posted either as new information or as an update on the blackboard. This posting or modification on the blackboard is referred to as an "event".

The system solves the route planning problem in the following manner. The user posts a mission specification on the control blackboard as an input. The system then attempts to trigger the knowledge sources based on that event. If one or more are triggered, only one is selected and executed generating one or more new events. These new events in turn cause other knowledge sources to be triggered, and perhaps execute. An "independent cooperation" among the knowledge sources ensues with knowledge sources triggering on events, executing their actions (one knowledge source per inference cycle), and posting the results of their actions on either blackboard. The route planning knowledge sources work out the details of the mission path and the control knowledge sources specify the order of route planning knowledge source execution. The solutions to both problems are incrementally generated on the blackboards and the final output of the Supervisor is a set of motion-commands pairs such as "goto x y z; do operation xxx".

At the heart of the control mechanism is the scheduling strategy used to choose the next KSAR from the agenda for execution. The scheduler plays a crucial role in influencing the outcome of the plans since it determines the planning behavior (i.e., the process by which the system generates the solution). Two strategies are used in the Supervisor, namely, successive refinement and Last-In-First-Out (LIFO) strategies. Successive refinement strategy directs the domain problem solution through its abstraction levels from the most abstract down to the most detailed so that knowledge sources at the higher abstraction levels have greater priority for execution. The LIFO strategy simply chooses the most recent KSAR for execution.

The Supervisor is currently designed to adopt one of the two strategies based on only one context parameter, namely, the time available to plan the mission which is part of the mission specification. If the allowable planning time is greater than a limit, successive refinement is chosen, otherwise the default LIFO strategy is used. This policy is adopted because results show that successive refinement strategy takes longer time to plan than LIFO and is thus less desirable when allowable planning time is low.

Currently, there are several issues that are not (yet) addressed by the system. Factors such as energy, risk, need for stealth and detection avoidance, etc., which are usually critical to a mission have not been considered. Moreover, the route planner assumes way-points are given, so that the problem reduces to one of sequencing them instead of deriving them from a priori environmental knowledge. Perhaps the more important questions relate to the architecture itself. Mayer [Ref. 15] points out several potential shortfalls with regard to blackboard architectures for mission control:

1. Lack of predictability, traceability and reliability of operation.
2. Inability to scope the effect of the data generated in the reasoning process.
3. High levels of communication traffic in a loosely coupled architecture.
4. Explosive increase in complexity of the "scheduler" as the number and complexity of the knowledge sources increases.
5. Inability to implement effective system level error detection and recovery procedures.

2. SITUATION BASED CONTROL ARCHITECTURE

This concept evolved from the blackboard architecture, apparently to effect a larger distribution of the knowledge based control components to different hardware processors, and to facilitate easier partitioning of knowledge sources along functional lines. A prototype Knowledge Based Control System (KBCS) for an AUV, based on this idea has been implemented at Texas A&M University to demonstrate its feasibility [Ref. 15].

The KBCS design revolves around the idea of a situation based architecture. This concept partitions the problem space into non-overlapping regions called situations. A situation encapsulates the rule sets, domain and declarative knowledge required to make the decisions, judgements, and actions required of the reasoning component in the corresponding part of the problem space. Situations can arise from either external or internal events, or combinations of the two. An entity called the Anticipator is responsible for monitoring the ongoing events and to "trigger" when certain scenarios such as "threat detection" or "mission replanning" occurs. When they trigger, the appropriate situation is retrieved from a situation database; the latter in turn triggers the actions of the various knowledge source components to deal with the situation.

In the prototype developed, the knowledge source components correspond roughly to the major functions of a submarine crew. Each component is hosted on a separate Symbolics 3640 machine and interconnected via an ethernet network. Five systems were developed, namely, the Skipper, the Navigator, the Engineer, the Diagnoser and the Facilitator. The Diagnoser is responsible for monitoring the other subsystems and to initiate recovery procedures when any failure occurs, while the Facilitator serves the inter-subsystem communication needs.

In a mission planning situation (or scenario), the Skipper, who is generally responsible for strategic and tactical planning, would request a path from one location to another from the Navigator. If a path can be found, a series of constrained paths will be tested. For example, the Skipper may order a path that will avoid standard shipping lanes. The Navigator will then search for a path that satisfies the constraints. When a path is returned to the Skipper, the Engineer is requested to perform a resource analysis for the path. The latter is essentially another constraint on the path (fuel) that must be considered before a final selection is made. After obtaining one or more constrained paths from the Navigator, The Skipper selects the mission plan that best satisfies the mission goals. All this time, the Facilitator serves the inter-subsystem communication needs.

Thus, unlike the blackboard approach where a single event triggers individual knowledge sources and where the knowledge sources reason independently, this approach relies on the anticipation of situations (based on a collation of one or more events) to trigger the cooperative actions of all the knowledge sour^es to deal with the task.

3. VALUE-DRIVEN HIERARCHICAL STRUCTURE

This approach to automating mission planning and control was first conceived at the University of New Hampshire under the NBS-UNH AUV program [Ref. 16]. The methodology emphasizes onboard planning and decision making in order to respond to unexpected events that may require major revisions in the mission route or plan, including decisions to omit some tasks originally planned for the mission.

The central contribution of the research is idea of a **value-driven** approach to decision making as opposed to rule-based decision making. In this approach, the critical mission factors such as vehicle survival, energy constraint, the time urgency for accomplishment of each task, the need for stealth, etc., are identified. For each factor, a value-priority indicating its criticality to the overall mission success is specified by the user. For instance, a value for the vehicles is used to assess the desirability of plan alternatives that may involve high risk to individual vehicles, or even the deliberate sacrifice of a vehicle, while a value of stealth for the mission would indicate the priority assigned to the avoidance of detection during the execution of the mission, and so forth. Each of the alternative plans is then evaluated in terms of the value criteria (or mission priorities) and the decision is completed simply by selecting the single alternative that shows the best projected score.

Except for resource-related constraints, all value-priorities are specified by the user. Resource constraints such as time and energy are treated differently since their usage (and thus, their value-priorities) can be varied as long as the total consumption of these resources does not exceed the supply. Using the latter condition, Lagrangean optimization techniques are applied to search for possible parameters (comprising the set of priorities as well as the sequence in which the mission tasks are to be executed) that gives optimal or near-optimal candidate plans with regard to the overall mission score. Each possible set of priorities is fed to a set of "outcome calculators" which provide the projected score for the plan.

The process ends with the selection when either a clearly satisfactory alternative has been identified, or when the available time for a decision has been exhausted. The output of the planner is the (macro-level) route for the vehicle and the tasks to be performed in sequence. The key to intelligent behavior in this approach lies in the correspondence of the valuative criteria with the higher-level objectives; the replanning that is performed whenever unanticipated circumstances occur enables the system to respond "intelligently".

C. PATH PLANNING ALGORITHMS

1. NAVIGATION FOR AN INTELLIGENT MOBILE ROBOT

An algorithm described by Crowley [Ref. 3] is designed for a mobile robot equipped with a rotating ultrasonic range sensor in a two-dimensional environment. This navigation system is based on a dynamically maintained model of the local environment, called the *composite local model*. The composite local model integrates information from the rotating range sensor, the robot's touch sensor, and a pre-learned global model as the robot moves through its environment. This work describes techniques for constructing a line segment description of the most recent sensor scan (the sensor model), and for integrating such descriptions to build up a model of the immediate environment (the composite local model). The estimated position of the robot is corrected by the difference in position between observed sensor signals and the corresponding symbols in the composite local model. Crowley also describes a learning technique where the robot develops a global model and a network of places. The network of places is used in global path planning, while the segments are recalled from the global model to assist in local path execution. The system is useful for navigation in a finite, pre-learned and man-made environment such as a house, office, or factory.

2. ROBOT NAVIGATION IN UNKNOWN TERRAIN USING LEARNED VISIBILITY GRAPHS

This algorithm, as described in [Ref. 4], deals with the problem of navigating an autonomous vehicle robot through unexplored terrain containing obstacles. A two-dimensional terrain, arbitrarily populated by disjoint convex polygonal obstacles, is assumed. The algorithm is proven to yield a convergent solution to each path of traversal. Initially, the terrain is explored using a rather primitive sensor, and the paths of traversal made to be near-optimal. The visibility graph that models the obstacle terrain is incrementally constructed by integrating the information about the paths traversed so far. At any stage of learning, the partially learned terrain model is represented as a learned visibility graph, and it is updated after each traversal. This work proves that the learned visibility graph converges to the visibility graph with a probability of one when the source and destination points are chosen randomly. Ultimately, the availability of the complete visibility graph enables the robot to plan globally optimal paths and also obviates further usage of sensors.

3. LEARNED NAVIGATION PATHS FOR A ROBOT IN UNEXPLORED TERRAIN

This algorithm is presented in [Ref. 5]. A method of robot navigation is proposed, which requires no pre-learned model, makes maximal use of available information, records and synthesizes information from multiple journeys, and contains concepts of learning that allow for continuous transition from local to global path optimum. Their model of the terrain consists of a spatial graph and a Voronoi diagram. Using acquired sensor data, two-dimensional polygonal boundaries are used to approximate the actual obstacle surfaces, free space for transit is correspondingly reduced, and additional nodes and edges are recorded based on path intersections and stop points. Navigation planning is gradually accelerated with experience since improved global map information minimizes the need for further sensor data acquisition. The method assumes that obstacle locations are unchanging, that navigation can be successfully conducted using two-dimensional projections, and that sensor information is precise.

4. AUTOMATIC PATH PLANNING FOR A MOBILE ROBOT USING A MIXED REPRESENTATION OF FREE SPACE

This algorithm, proposed in [Ref. 6], uses a mixed representation of free space in terms of two shape primitives: generalized cones and convex polygons. Given a set of polygonal obstacles in space, the planning algorithm first identifies the neighborhood relations among obstacles and uses these relations to localize the influence of obstacles on free space description, and then locates critical "channels" and "passage regions" in the free space. The free space is then decomposed into non-overlapping geometric-shaped primitives where the channels are represented as generalized cones similar to those introduced by Brooks [Ref. 7]. The passage regions are represented as convex polygons. Based on this mixed representation of free space, the planning algorithm uses two different strategies to path plan trajectories inside the channels and passage regions.

5. HEURISTIC TWO-DIMENSIONAL NAVIGATION ON ROUGH TERRAIN WITH OBSTACLES

The algorithm is described in [Ref. 8]. It is designed for autonomous land vehicle navigation in situations where no a priori terrain information is available. The method models the terrain as a regular two-dimensional grid system with height information stored at each cell. Thus, the path search uses the traditional eight-neighbor search strategy. The path search process is guided by a set of heuristics intended to mimic closely what a human navigator would do in similar circumstances. In the implementation, these heuristics are represented as mathematical functions. For instance, the heuristic to "move toward the destination whenever possible" is captured in an estimation function, while the rule "try not to visit the positions already explored" is represented by a path-marking function. A significant contribution of the work is in the area of obstacle clearance; conceptually, whenever obstacles are encountered by the vehicle, it is made to detour along the periphery of the obstacle until the latter is cleared. Results show that for flat or moderately sloped terrain, the method provides an almost optimal path (in terms of energy required), while highly sloped terrain yields reasonable paths. It is also highly efficient in the usage of computer CPU and memory resources. This approach forms the basis of the Heuristic search developed in this thesis for three-dimensional underwater environments.

D. SUMMARY

This chapter provides a broad survey of research work that has been done in the area of mission planning and control for Autonomous Underwater Vehicles, and in the area of path-planning in general. Three different system architectures for mission planning and control are examined - Blackboard Based systems, the Situation Based Control Architecture and the Value-driven Hierarchical Architecture. In the area of path-planning, previous research has concentrated on two-dimensional path-planning with little attention given to three-dimensional problems. In particular, the methods surveyed are targeted for land-based vehicular and robotic applications. Published work on path-planning for underwater environments is scarce.

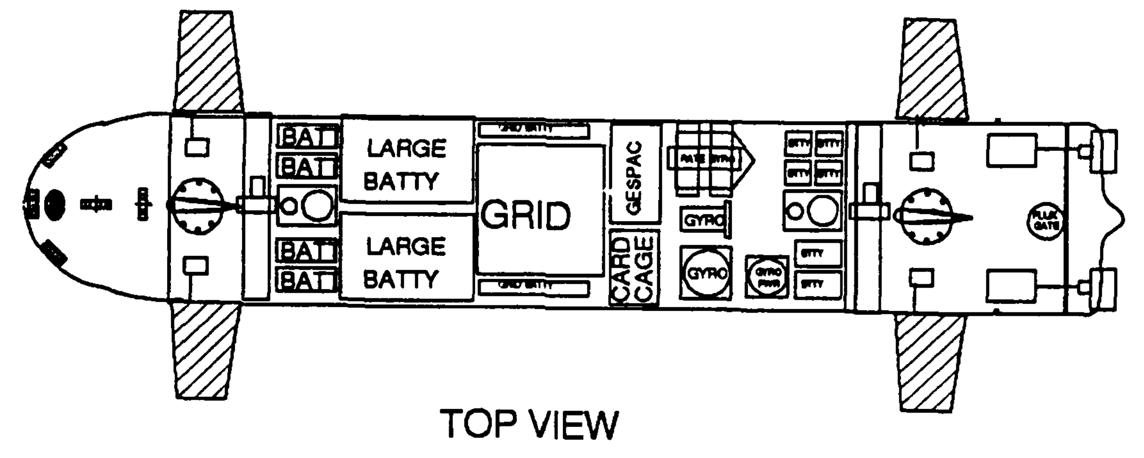
III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

This thesis further advances the evolutionary development of an automated mission planning and control system for the NPS-AUV program. A key feature of the mission planning expert system developed is its ability to select an appropriate path-search strategy for a particular mission. The output of the system is a detailed path specification that fulfills the mission requirements and constraints. The path is constructed using one of three alternative path-search methods, namely, A*, Best-first, or Heuristic search. In particular, Heuristic search is proposed as a new path-search strategy for autonomous vehicles in three-dimensional underwater environments.

B. NPS AUV PHYSICAL CHARACTERISTICS

The current vehicle is called the "NPS Model 2 AUV" and is based to a considerable degree on the earlier, smaller Model 1 AUV described in [Ref. 17]. The overall appearance and layout of the Model 2 AUV is shown in Figure 3.1. As can be seen, the vehicle has a rectangular cross-section and is furnished with four forward control surfaces and four aft control surfaces, as well as four tunnel thrusters. These thrusters, combined with the two aft screws, provide the vehicle with active control



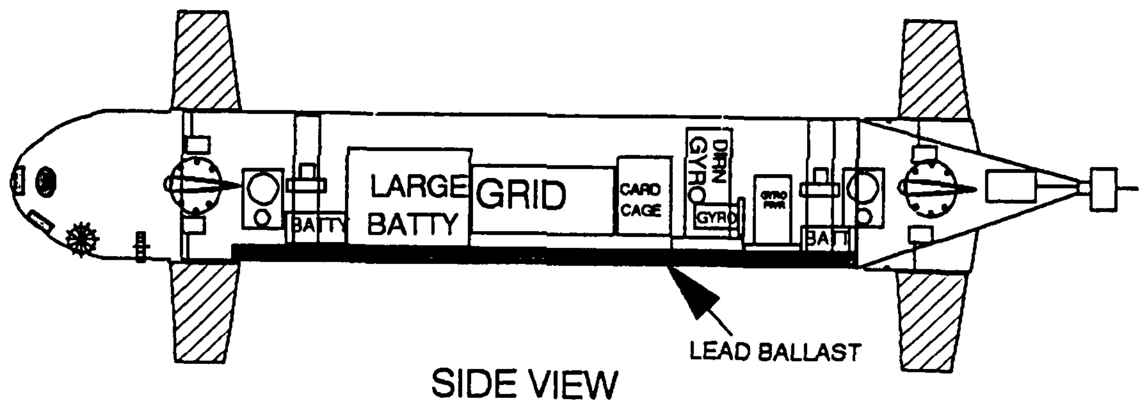


Figure 3.1 NPS Model 2 AUV

of five degrees of freedom in a low speed ***hovering mode***, with only the roll degree of freedom being passively controlled. When the vehicle is operated in its higher-speed ***transit mode***, thrusters are not used and all six degrees of freedom are actively controlled using the aft main screws for propulsion and hydrodynamic forces on the control surfaces to achieve commanded rotational rates in roll, pitch, and yaw. The total weight of the vehicle is 387 lbs and its length is 93 inches.

As can also be seen from Figure 3.1, the Model 2 AUV is battery powered and contains two on-board computers, a Gridcase 80386 based laptop computer, and a Gespac 68030 based real-time control computer. The Gespac computer is furnished with depth and speed sensors, a complete suite of inertial sensors (3 rate gyros, 3 accelerometers, vertical gyro, directional gyro, and flux-gate compass), and a sonar system for obstacle avoidance and bottom sounding. As indicated in the figure, the latter system consists of four fixed-base pencil-beam sonar rangers mounted in a flooded fiberglass nose cone. One sonar beam looks downward at 45 degrees, another forward, and the other two are aimed diagonally to the right and left of the forward looking beam. It is currently anticipated that the Gridcase computer will be programmed in Common Lisp and will run under the MS-DOS operating system while the Gespac computer will be programmed in C and will run under OS-9 [Ref. 18].

C. CONTROL SYSTEM ARCHITECTURE

Figure 3.2 shows the current system architecture which depicts how the vehicle mission planning, mission control, and vehicle control functions are divided. Although the hierarchical structure is inherited from previous thesis work [Ref. 1, 2], some major re-organization and enhancements has been made. In particular, the previous mission selection supervisor has been replaced by a mission planning expert system at the Mission Planning level, which is the focus of this thesis.

The figure is subject to multiple interpretations depending upon what computers host the software. Currently, at the time of writing this thesis, the Mission Planning and Mission Control levels reside in a Symbolics 3675 Lisp machine, while the Vehicle Control level and the simulation of the vehicle and environment are implemented in a Silicon Graphics Iris 4D/GT graphics workstation. Figure 3.3 shows a typical image from a simulated mission using the latter system.

The next stage in the development of the software system of Figure 3.2 will involve downloading of Mission Control software into a laboratory duplicate of the on-board Gridcase computer. Simulated missions will then be run in the laboratory with the SGI graphics workstation function unchanged from its role in the current stage of software development. It is expected that this mode of operation will represent a stable configuration for mission planning for the Model 2 AUV. That is, it is anticipated that on-board mission control software will in general be mission dependent and that, before being installed in the vehicle, all such software will be tested initially in the laboratory on a duplicate Gridcase computer.

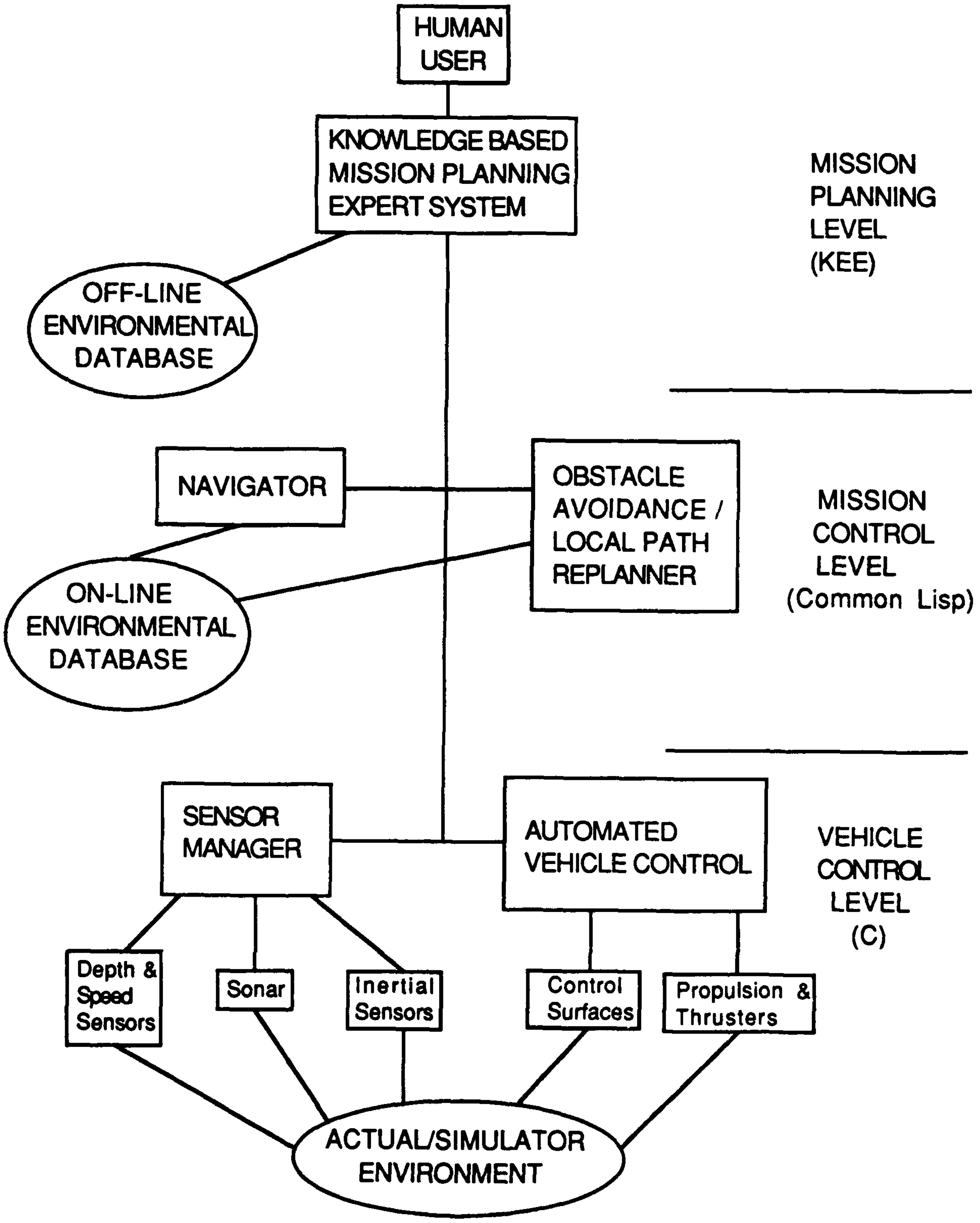

HUMAN USER
KNOWLEDGE BASED MISSION PLANNING EXPERT SYSTEM
MISSION PLANNING LEVEL (KEE)
OFF-LINE ENVIRONMENTAL DATABASE
NAVIGATOR
OBSTACLE AVOIDANCE / LOCAL PATH REPLANNER
MISSION CONTROL LEVEL (Common Lisp)
ON-LINE ENVIRONMENTAL DATABASE
SENSOR MANAGER
AUTOMATED VEHICLE CONTROL
VEHICLE CONTROL LEVEL (C)
Depth & Speed Sensors
Sonar
Inertial Sensors
Control Surfaces
Propulsion & Thrusters
ACTUAL/SIMULATOR ENVIRONMENT

Figure 3.2 Control System Software Architecture

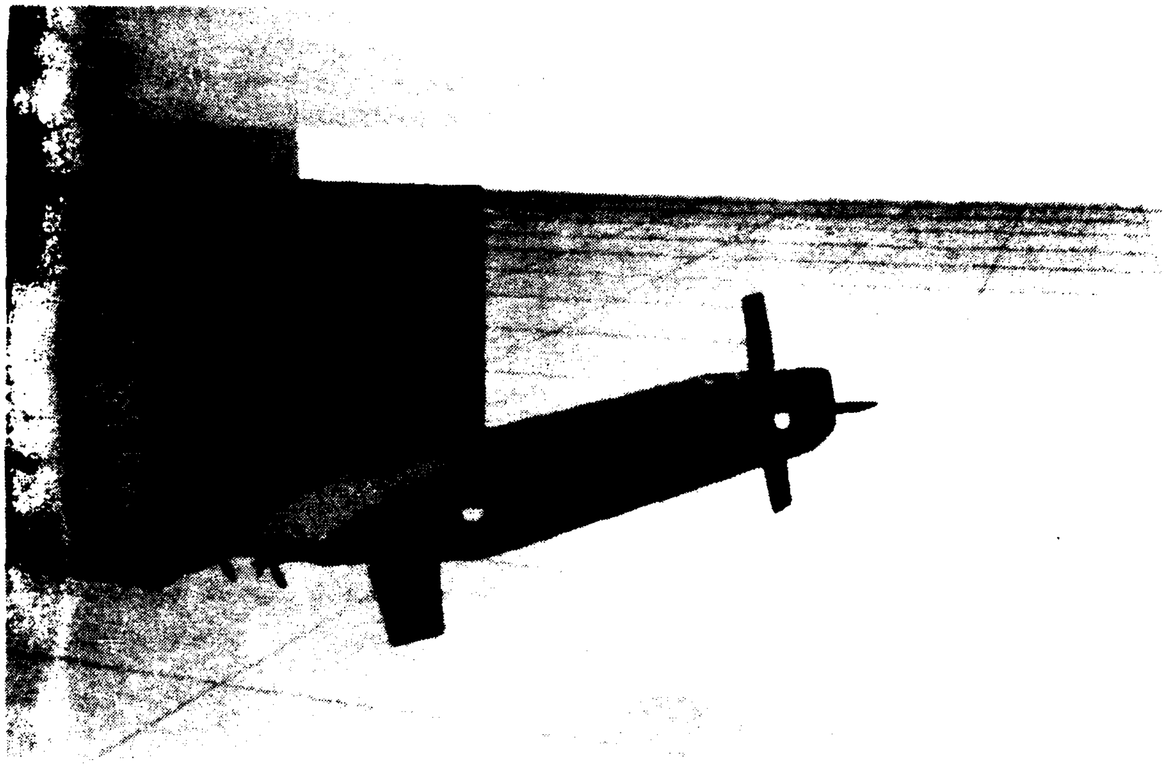


Figure 3.3 Graphical Display of Simulated AUV and Test Pool

The third interpretation of Figure 3.2 is that Mission Planning software will be hosted on a smaller delivery system Lisp machine, currently a Texas Instruments Micro-Explorer. This system will be portable and will be part of the AUV pre-launch checkout and initialization system. In this case, Mission Control software will be automatically generated and downloaded to the AUV just before launch. In this configuration, Vehicle Control level software, implemented in C, will have previously been installed in the Gespac real-time control computer. It is expected that the latter software will be relatively stable and generally *not* mission dependent.

At the time of this writing, only the first interpretation of the software system for the Model 2 AUV is fully operational. Moreover, much remains to be done to further expand this system to better support meaningful AUV operations. In parallel with this activity, work is also under way to realize the second and third interpretations. This thesis, however, is confined to the Mission Planning software at the Mission Planning level.

D. MISSIONS

The Defense Advanced Research Projects Agency (DARPA) has identified over 70 military missions especially suited for AUV execution [Ref. 2, 20]. The current stage of development of the system considers only a minute subset of these, and classifies them under four categories: routine, charting, covert, and intelligence missions. Further, although multi-task missions are common, this study assumes only single-task missions with the following generic three-phase structure: transit from start to a goal location, perform task upon reaching destination, and then return to the start location. The high-level mission specifications considered are: available planning time, mission depth, mission threat level (stealth requirement), and mission range (computer resource requirements). Other constraints implicit in all missions include obstacle clearance and collision avoidance.

In order to validate the performance of the expert system and the path-search strategies, a generic test mission template, called ***Transit Pool***, has been defined in which the area of operation is the proposed test site for the actual vehicle, namely, the NPS swimming pool. Given the start and goal locations in the pool and the mission specification, this mission requires the system to construct a detailed path using an appropriate path-search method; the vehicle is then required to navigate itself in accordance to the path derived, and to maneuver around obstacles placed in its path. However, prior to actual in-water tests, several simulated executions under varying mission specifications must first be performed using the AUV laboratory testbed.

E. PATH PLANNING ASSUMPTIONS

1. ENVIRONMENT MODEL

In this study, the environment is the NPS swimming pool. The latter is modelled by a 3-dimensional Cartesian coordinate system with the X-Y plane parallel to the surface of the pool and the Z-axis pointing towards increasing depth of pool (Figure 3.4). Each unit of the X and Y-coordinate is 70 inches (corresponding approximately to the length of the vehicle), while each unit of the Z-coordinate is 10 inches (corresponding approximately to the height of the vehicle). Henceforth, the units shall be referred to as the ***grid units***, and the coordinate system as the ***path-planning coordinate system***, the ***grid system*** or simply the grid.

In this model, a three-dimensional ***unit cell*** with unit length on all sides is defined. Note that this cell is not a cube because one unit Z-coordinate is shorter than one unit X or Y-coordinate. This then becomes the resolution of the environment as all locations are resolved to a unit cell at the specified (x,y,z) coordinate.

Another assumption is that the underwater environment is homogeneous all round; that is, changes in pressure, temperature, density and viscosity of fluid, which affect the resistance to vehicle movement are not modelled. Thus, the energy cost per unit distance is constant everywhere in the environment. Note, however, that

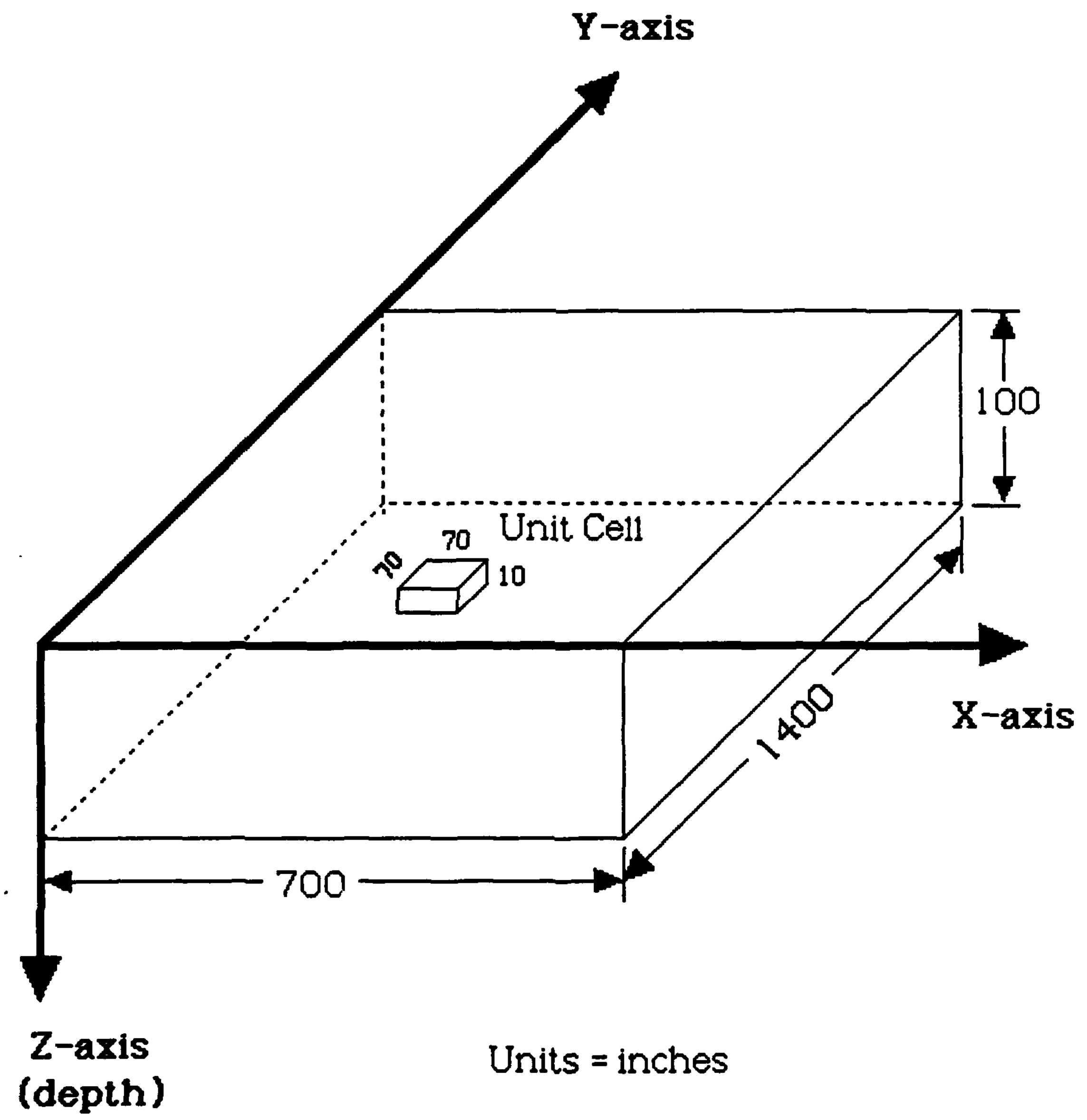


Figure 3.4 Environment Model

the cost rate for vertical movements and that for horizontal movements may still be different, since these are vehicle-related rather than environment-related constraints.

2. OBSTACLE MODEL

Only static obstacles are modelled, although the heuristic search algorithm can be extended to handle dynamic obstacles [Ref. 8]. The smallest size of an obstacle is a unit cell, and larger objects are approximated by a lego-style assembly of multiple unit cells.

One problem inherent in vehicle dynamics which is addressed in the obstacle model is that there is always a finite distance required to bring a moving vehicle to a halt, as well as some finite radial distance associated with any vehicle turns. For instance, it would not be realistic to expect the vehicle to head straight for an object and then make a sharp dive or turn without hitting it. To circumvent the problem, a concept called ***obstacle-growing*** is adopted [Ref. 21]. The obstacle-growing process increases the size of the obstacles by one unit cell all round its periphery. Thus a ***virtual obstacle*** is created which is larger than the real obstacle. In the subsequent discussions and figures shown, real obstacles are implied, unless explicitly stated.

3. VEHICLE MODEL

The following conceptual model of the vehicle is assumed:

1. The size of the vehicle is approximately the unit cell size of the environment.
2. The vehicle remembers all the places it has visited.

3. It expends a certain amount of energy whenever it moves from its current location to a new position.
4. It tracks its own position with absolute accuracy.

4. SENSOR MODEL

This sensor model is not applicable to A* and Best-first search methods, since they require complete a priori information. For Heuristic search, the sensors must facilitate the building of a model of its immediate surrounding environment. More accurately, it is assumed that the onboard vehicle sensors are able to sense the surrounding environment defined by a rectangular boxed region with dimensions (5 x 5 x 5) grid units, with the vehicle at the center. Note that it would not be sufficient for the dimensions to be (3 x 3 x 3) grid units; this is because the vehicle must be able to sense at least two grid units all around itself in order to detect a *virtual* obstacle.

F. SIMULATION FACILITIES

As mentioned in Chapter I, a laboratory testbed environment has been developed as a result of previous thesis work. This testbed is configured from three separate systems: a Symbolics 3675 LISP machine, a Symbolics Color Monitor and a Silicon Graphics IRIS 4D/70GT graphics workstation. The LISP machine together with the Symbolics Color Monitor is set up as the Mission Planning Workstation; the former hosts the Mission Planning software, while the latter is used to display the derived path (as well as the actual path during execution phase) in two-dimensional plan and side-elevation views of the NPS pool. The IRIS graphics workstation is for 3-D visualization of the vehicle and environment during mission execution. The

Mission Planning Workstation and the IRIS workstation communicate via an ethernet network using TCP/IP protocol.

In parallel with the mainstream work of this thesis, the C-code for the IRIS graphics has been significantly enhanced. First, the code has been modularized to facilitate easier maintenance in the future. Secondly, the display of the swimming pool has been modified to reflect the actual dimensions and, in particular, the tapering depth of the pool is shown. Thirdly, all objects in the display such as the pool and the vehicle itself has been converted to an Object File Format (OFF) [Ref. 22], again to facilitate easier modifications in the future. Lastly, the new NPS Model 2 AUV has been added to the display. These changes were necessary not only as an upgrade of the simulator, but also to bring it on par with the current status of the overall project.

G. SUMMARY

This chapter discusses in detail the problems addressed by this thesis. The current vehicle characteristics are described; in particular, the planned incremental realization of its control architecture is highlighted. The basic underlying assumptions for the development of both the Mission Planning Expert System and the path-search strategies are also listed and described in detail. They include assumptions concerning the type of missions considered, the environment and obstacle models, and the vehicle and the sensor models. Finally, the role of the laboratory testbed used in this thesis is explained.

IV. MISSION PLANNING EXPERT SYSTEM

A. SOFTWARE ARCHITECTURE OVERVIEW

The Mission Planning Expert System (MPES) is physically hosted on a standalone Symbolics 3675 Lisp machine. Conceptually, it resides at the mission planning level (Figure 3.2). The internal structure of the system, as shown in Figure 4.1, is essentially hierarchical and is patterned after the progressive phases of a mission, namely, the initiation, planning, construction, and execution phases. The system has been developed entirely in the KEE expert system shell [Ref. 23]; the corresponding KEE knowledge base is shown in Figure 4.2.

In this architecture, control of the planning operation is centralized at the top-level Mission Planning Controller (analogous to a real-life Mission Commander) which is designed to oversee the entire mission and to enforce an orderly transition from one phase to another. In addition to the Controller, there are four other distinct elements or ***role-players*** in the system corresponding to the four main mission phases. They are the Mission Receiver, the Mission Planner, the Mission Constructor and the Mission Executor. Of these, the latter three are charged with the core mission planning and tasks, and are collectively referred to as the ***Mission Planning Agents***. Communication between individual elements is effected by means of formal ***documents*** realized as ***KEE units*** [Ref. 23].

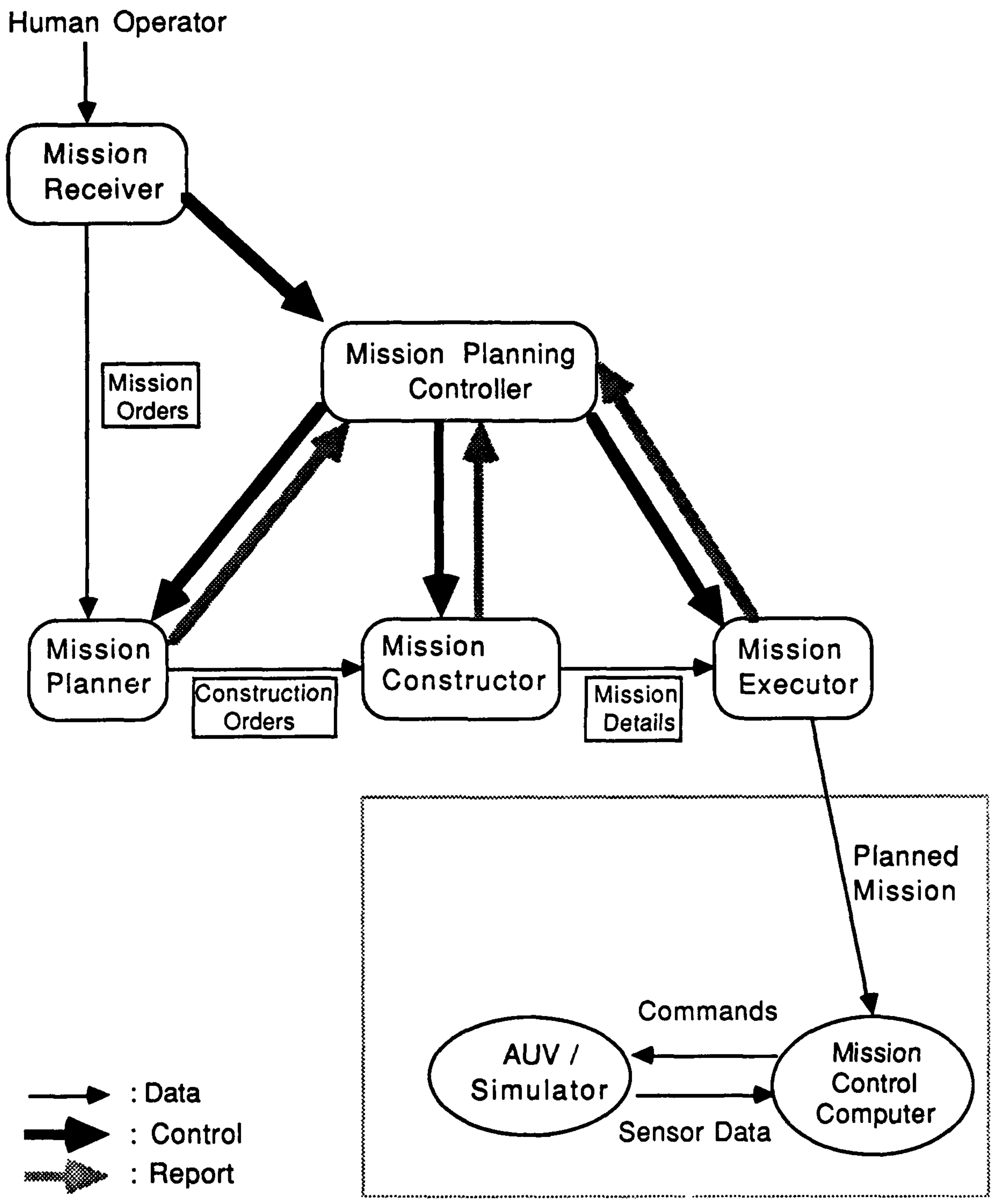


Figure 4.1 Structure of Mission Planning Expert System

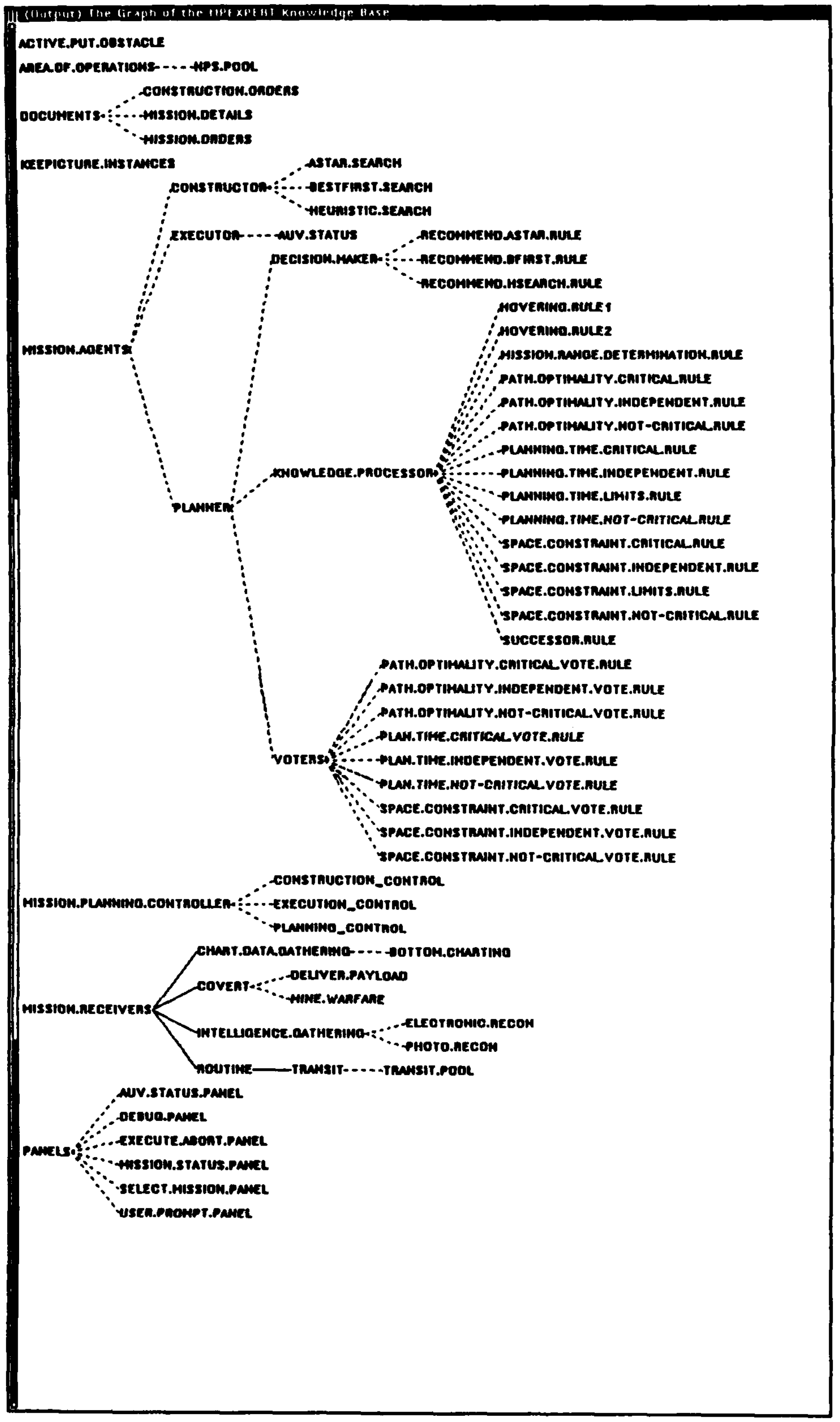

(Output) The Graph of the MPEXPERT Knowledge Base
ACTIVE.PUT.OBSTACLE
AREA.OF.OPERATIONS
NPS.POOL
DOCUMENTS
CONSTRUCTION.ORDERS
MISSION.DETAILS
MISSION.ORDERS
KEEPICTURE.INSTANCES
MISSION.AGENTS
CONSTRUCTOR
ASTAR.SEARCH
BESTFIRST.SEARCH
HEURISTIC.SEARCH
EXECUTOR
AUV.STATUS
PLANNER
DECISION.MAKER
RECOMMEND.ASTAR.RULE
RECOMMEND.BFIRST.RULE
RECOMMEND.HSEARCH.RULE
KNOWLEDGE.PROCESSOR
HOVERING.RULE1
HOVERING.RULE2
MISSION.RANGE.DETERMINATION.RULE
PATH.OPTIMALITY.CRITICAL.RULE
PATH.OPTIMALITY.INDEPENDENT.RULE
PATH.OPTIMALITY.NOT-CRITICAL.RULE
PLANNING.TIME.CRITICAL.RULE
PLANNING.TIME.INDEPENDENT.RULE
PLANNING.TIME.LIMITS.RULE
PLANNING.TIME.NOT-CRITICAL.RULE
SPACE.CONSTRAINT.CRITICAL.RULE
SPACE.CONSTRAINT.INDEPENDENT.RULE
SPACE.CONSTRAINT.LIMITS.RULE
SPACE.CONSTRAINT.NOT-CRITICAL.RULE
SUCCESSOR.RULE
VOTERS
PATH.OPTIMALITY.CRITICAL.VOTE.RULE
PATH.OPTIMALITY.INDEPENDENT.VOTE.RULE
PATH.OPTIMALITY.NOT-CRITICAL.VOTE.RULE
PLAN.TIME.CRITICAL.VOTE.RULE
PLAN.TIME.INDEPENDENT.VOTE.RULE
PLAN.TIME.NOT-CRITICAL.VOTE.RULE
SPACE.CONSTRAINT.CRITICAL.VOTE.RULE
SPACE.CONSTRAINT.INDEPENDENT.VOTE.RULE
SPACE.CONSTRAINT.NOT-CRITICAL.VOTE.RULE
MISSION.PLANNING.CONTROLLER
CONSTRUCTION_CONTROL
EXECUTION_CONTROL
PLANNING_CONTROL
MISSION.RECEIVERS
CHART.DATA.GATHERING
BOTTOM.CHARTING
COVERT
DELIVER.PAYLOAD
MINE.WARFARE
INTELLIGENCE.GATHERING
ELECTRONIC.RECON
PHOTO.RECON
ROUTINE
TRANSIT
TRANSIT.POOL
PANELS
AUV.STATUS.PANEL
DEBUG.PANEL
EXECUTE.ABORT.PANEL
MISSION.STATUS.PANEL
SELECT.MISSION.PANEL
USER.PROMPT.PANEL

Figure 4.2 KEE Knowledge Base for Mission Planning Expert System

The Mission Receiver is solely responsible for interacting with the User concerning the mission orders (specifications). This is done, during the initiation phase, via the Mission Planning Workstation described in Section D of this chapter. When the orders have been completed, control is passed to the Mission Controller which then initiates the actions of the Mission Planning Agents. The first of these agents, the Mission Planner, is a key element in the system; it embodies the essential "intelligence" or "knowledge" for deciding the most appropriate path-search strategy to be used, based on the current mission constraints. The Planner's decision is currently based on three main parameters, the range of the mission (determined from the start and goal coordinates), the time available for planning the mission, and the threat level of the mission. The design of the rule-based Planner is described in greater detail in Section C of this chapter. Once the path-search strategy has been decided, the decision is registered in the ***Construction Orders*** document and passed to the Mission Constructor.

The Mission Constructor is currently equipped with three search methods or tools, as shown in Figure 4.2, under the Mission Constructor. They are the A* search, Best-first search [Ref. 9, 10], and Heuristic search methods, all of which perform three-dimensional grid-based search. Among these methods, only A* is guaranteed to produce an optimal path. Best-first search produces generally good (but not always optimal) paths with less time and space than A*, while heuristic search provides only reasonable paths, but does so very quickly. These generalizations are justified based on the results of a series of tests conducted to evaluate their performance (described in Chapter VI).

Upon initiation by the Controller, the Mission Constructor proceeds to construct the detailed path using the selected search method and the off-line environmental database. In this process, it ensures that all operational requirements with regard to threat avoidance, operating depth, etc., as well as any restrictions in vehicle motion, are considered. The output from the Constructor is the ***Mission Details*** document containing the low-level execution details of the mission. Currently, it contains the path definition (which is a series of way-points in the path) and the activity to be performed upon reaching the target/goal location. This document is passed to the Mission Executor for the next stage of the mission - the execution phase.

The role of the Mission Executor is to interface between the MPES at the Planning level and the on-board computers at the Mission Control level. In actual missions, it downloads the planned Mission Details to the AUV for execution, when commanded by the Controller. In the laboratory setup, however, it is designed to drive the AUV simulator running on the SGI graphics workstation; that is, it emulates the Mission Control level function by monitoring and controlling the simulated vehicle as it navigates along the prescribed path.

B. REPRESENTATION OVERVIEW

As mentioned, the MPES is implemented using the KEE expert system shell. Thus, the five distinct role-players in the system as well as the three documents shown in Figure 4.1, are implemented as KEE units. A KEE unit is a basic entity in the KEE environment. It is a block of Lisp code similar to an instantiation of a Common Lisp ***class*** [Ref. 24], but with added functionality. Specifically, unit slots in KEE can hold procedures (or functions) called ***methods***, and not just attributes or components as in Common Lisp. This feature of KEE produces a more explicit encapsulation of methods with objects than is provided for by CLOS, the Common Lisp object standard [Ref. 25].

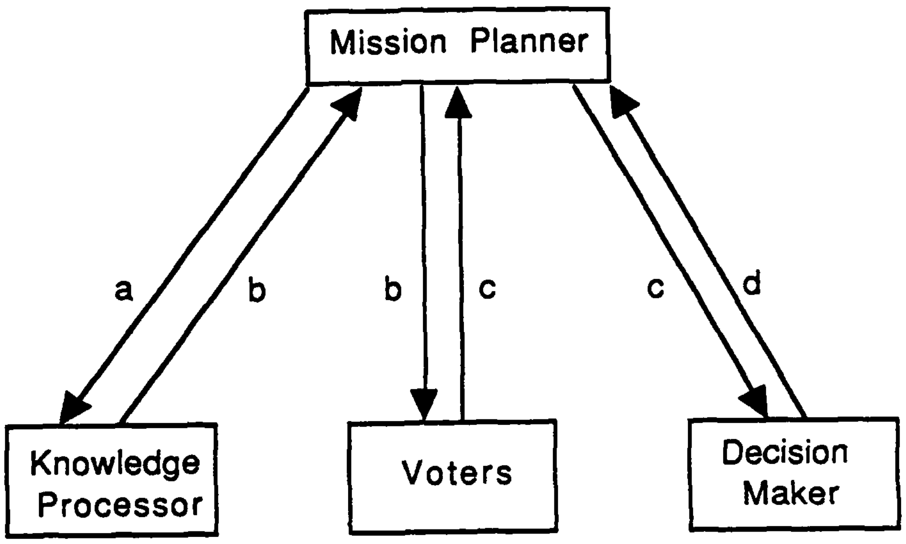
KEE units which make use of methods in their slots are procedural or method-based units. KEE units, however, can also be rule-based; these are units that contain rules rather than methods. Rule-based units are employed for functions which are not suited for an algorithmic solution. Planning, for instance, is a rather unstructured and poorly understood problem -- and in particular, mission planning. The same is true for a generalized mission controller which makes decisions based on dynamically changing situations; for instance, decisions to skip a mission phase, or to abort the current mission phase in order to begin re-planning due to unforseen circumstances - such decisions usually involve a great deal of judgement, and the reasoning and analysis process is generally unstructured. Employing rule-based reasoning in these areas also facilitates understanding by human experts. For these reasons, the Mission Planning Agent and the Mission Controller are implemented as rule-based units.

On the other hand, the Mission Receiver, Constructor and Executor are implemented as method-based KEE units because they execute well-defined tasks with completely defined input and output. They possess slots containing procedures which perform their tasks. Lastly, the three documents are simple units with slots meant only for data storage.

C. THE MISSION PLANNER

The central role of the Mission Planner agent is to decide which of the available search methods would best fulfill the given mission requirements. In order to reach this decision, it works with three ***specialists***: the Knowledge Processor, the Voters, and the Decision Maker, which are realized as three different rule sets operating under the Mission Planner. The Mission Planner controls the operations of these three specialists by providing information to and receiving processed information from them sequentially.

The interactions between the planner and the specialists are shown in Figure 4.3. First, the Mission Planner makes the Mission Orders available to the Knowledge Processor and initiates its operation. The latter processes the high-level information and transforms them to "intermediate knowledge" that is readily understood by the



a: Mission Orders

b: Standard Intermediate Knowledge

c: Voting Values with Signature

d: Decisions

Figure 4.3 Mission Planner and Three Specialists

Voters. When this processing is completed, the Mission Planner receives the "intermediate knowledge" from the Knowledge Processor and passes it to the Voters. The Voters correspond to the intermediate knowledge - each Voter provides voting values to the search methods according to its strength or weakness in the relevant area of the path-search process. Upon receipt of the voting values from the Voters, the Mission Planner initiates the operation of the Decision Maker. The Decision Maker then makes a decision based on the voting values and the "credibility" of the individual voters, and sends its decision to the Mission Planner. Finally, the Mission Planner generates the Construction Orders on the basis of that decision. The following sub-sections describe the implementation of the three specialists in greater detail.

1. THE KNOWLEDGE PROCESSOR

As its name suggests, the Knowledge Processor processes knowledge - specifically, it transforms the high-level information contained in the Mission Orders to "intermediate knowledge" that is understood by the Voters. Intermediate knowledge here, refers to the degree of criticality associated with the factors pertinent to path planning, such as the time and space constraints, and the optimality of the path required. The transformation is done in two stages: first, it processes the Mission Orders, and then it generates the intermediate knowledge on the basis of the first step. A total of 15 rules are used - three for the first stage and the rest for the second stage.

a. Processing the Mission Orders

The following three rules are used to first process the Mission Orders: "Mission.Range.Rule", "Space.Constraint.Limits.Rule", and "Planning.Time.Limits.Rule". The first of these three rules is responsible for estimating the mission range (or distance). This estimate is needed in order to determine the computing memory space and time needed for the whole mission, even though an exact mission distance is not available before completing a path to the goal. Thus, a gross estimate is obtained by simply taking the horizontal straight-line distance between the start and the goal positions before planning a path.

The second, the "Space.Constraint.Limits.Rule", determines the upper limits and the lower limits for computer memory space requirements. This information is used to determine whether the currently available computer space is sufficient to plan a mission. Because the greatest requirement for computer space is generated by the Mission Constructor, the overall space requirements are based entirely on its needs. Moreover, since the Constructor has three search methods at its disposal, the most complex of these, A*[6], is used to estimate the needed space. Based on experiments with the Constructor, the branching factor for A* search averages 1.45, and approximately 14 units of storage are needed at each node of the search tree. Consequently, the estimated space requirement (ESR) for A* search is given by

$$ESR = 14 * (1.45)^{D} \tag{4.1}$$

where D is the mission distance measured in grid units. Since this relationship is approximate, before the value for ESR is used by the mission planner, it is transformed into an upper and lower bound as follows:

$$UESR = 2 * ESR \tag{4.2}$$

$$LESR = 0.5 * ESR \tag{4.3}$$

The "Planning.Time.Limits.Rule" performs a task similar to "Space.Constraint.Limits.Rule". This rule calculates ETR (Estimated Time Requirement) using the following equation:

$$ETR = 2.3 * 10^{-3} * (2.1)^{D} \tag{4.4}$$

This equation is derived from the observation that search time is proportional to the size of the search tree, and that the size of the tree is mainly determined by the maximum width of the tree. Thus, the same type of equation as that for ESR is introduced to calculate ETR, and the effective branching factor, 2.1, is again measured from experiments. As for space constraints, the value obtained from Eq. 4.4 is transformed into lower and upper bounds by multiplying by a factor of 0.5 and 2.0 respectively.

b. Generating Intermediate Knowledge

The upper and the lower bounds on time set by this calculation are used by the "Planning.Time.Critical.Rule", "Planning.Time.Not-Critical.Rule", and "Planning.Time.Independent.Rule". Depending on the available time given through the Mission Orders, one of these rules is fired. If the available time is less than the lower bound, then the "Planning.Time.Critical.Rule" is fired. In this case, "Planning time is critical", a standard form of intermediate knowledge, is given to the Mission Planner. This is actually done by saving "critical" into the "planning-time" slot of the Mission Planner unit. If the available time is larger than the upper bound, then the "Planning.Time.Independent.Rule" is fired, and the value "independent" is saved into the "planning-time" slot. Otherwise, the "Planning.Time.Not-Critical.Rule" is fired, and this rule puts "not-critical" into the "planning-time" slot.

Similarly, the "Space.Constraint.Critical.Rule", "Space.Constraint.Not-Critical.Rule", and "Space.Constraint.Independent.Rule" utilize the upper and lower bounds on space to generate the standard intermediate knowledge about the space constraint. Depending on the comparison result, the value "critical", "not-critical", or "independent" is saved into the "space-constraint" slot of the Mission Planner.

The rules relating to path optimality, "Path.Optimality.Critical.Rule", "Path.Optimality.Not-Critical.Rule", and "Path.Optimality.Independent.Rule", generate the intermediate knowledge for the Voters from the threat information in the Mission Orders. Depending on whether the threat level is hostile, neutral, or friendly, the "path-optimality" slot of the Mission Planner is set to "critical", "not-critical", or "independent", respectively.

The "Successor.Rule" checks the mission threat level and, when the threat is hostile, puts "shallow successor not allowed" into the "successor-mode" slot of the Mission Planner in order to keep the Mission Constructor from considering a shallower path segment than the mission depth during path construction. However, this successor information is not part of the intermediate knowledge and is moved directly to the Construction Orders by the Mission Planner when the planning phase is completed.

Two rules relating to the AUV hovering mode, "Hovering.Rule1" and "Hovering.Rule2", put a proper value into the "vertical-successor" slot of the Mission Planner depending on the information as to whether the hovering mode is allowed or not as specified by the user through the Mission Orders. Like the successor information set by the "Successor.Rule", the "vertical-successor" slot information is directly transferred to the Construction Orders without further processing.

2. THE VOTERS

The Voters, another of three specialists under the Mission Planner, mimic a group of people casting ballots based on their own judgements. As shown in Figure 4.2, nine voting rules are implemented. Those voting rules which match with the intermediate knowledge generate ***favor*** values, which lie in the interval 0 to 1. The Voters also append their "signatures" to the "favor" values so that the credibility or the importance of the "favor" values can be weighted by the Decision Maker, the last specialist under the Mission Planner. Each rule is composed of one condition in the

LHS (left hand side) of the rule and three actions in the RHS (right hand side). When the LHS condition matches with one of the assertions in the intermediate knowledge, the RHS three actions generate three "favor" values for the three search methods. Therefore, these nine rules act as a favor value look-up table as well as a reader of the table. The currently implemented favor values are shown in Table 4.1. These values have been carefully selected based on simulation experience to produce reasonable results for various cases. Because of the rule-based approach, whenever a new table entry is introduced, a new rule can be simply added without affecting other voting rules.

Table 4.1 Favor Values used by Voting Rules

		A*	Best-first	Heuristic
Planning Time	Critical	0.2	0.1	1.0
	Not Critical	1.0	1.0	1.0
	Independent	1.0	0.5	0.5
Space Constraint	Critical	0.3	0.3	1.0
	Not Critical	0.7	0.7	1.0
	Independent	0.9	0.9	1.0
Path Constraint	Critical	1.0	0.5	0.5
	Not Critical	1.0	0.7	0.6
	Independent	1.0	1.0	1.0

3. THE DECISION MAKER

The Decision Maker, the last specialist under the Mission Planner, makes a recommendation using the favor values. It can discriminate among favor values based on the signatures provided with them. Currently, no favor values are weighted differently because the Decision Maker works satisfactorily without different weighting factors. When the operation of the Decision Maker is initiated by the Mission Planner, the Decision Maker calculates its own final scores of three search tools by adding up the favor values. After the final scores are calculated, three rules become active to select the search tool which gets the highest final score. Basically, the three rules oppose each other until the highest score is set by the rule which matches with the highest score among them. The LHS rule compares the score of a specific search tool and the highest score which is in temporary storage in the Decision Maker. If the matched score is higher than the highest score in the Decision Maker, then the RHS rule changes the highest score to the matched score and declares the search tool as the winner. Therefore, when rule firing is terminated, the highest score as well as the winner is recorded in the Mission Planner unit. Because of this approach, when an additional search tool is added into the Mission Constructor, another rule can be simply added without modifying the existing rules. Note that although this decision making is internally performed in two phases in the Decision Maker, the Mission Planner simply sends one message, "Start" to the Decision Maker. The method (procedure) execution and the rule firings are sequentially performed by the Decision Maker itself.

D. MISSION PLANNING WORKSTATION

1. PURPOSE AND DESIGN CONSIDERATIONS

The Mission Planning Workstation is configured using a Symbolics 3675 LISP machine and an external Symbolics Color Monitor, as mentioned in Chapter III. Its purpose is to provide the user with an interactive, easy-to-operate, display workstation from which to plan and monitor the progress of a mission. This is achieved through the provision of several ***image panels*** for:

1. Selecting a mission.
2. Entering the parameters and data for the selected mission.
3. Pre-viewing the detailed plans for the mission and, in particular, the path.
4. Monitoring the current mission status and the AUV operating status during execution.

Except for (3), all the panels are developed on the LISP machine using the KEE graphics facility. In order to produce an easy-to-operate system, two principles were observed in the design: firstly, the user is prompted at each step, and secondly, in order to avoid "information overload", all data that is irrelevant to a specific phase of the mission is either inhibited from display or hidden. A preview of the detailed plans is facilitated by the display of a two-dimensional representation of the path on the Symbolics Color Monitor.

2. AN ILLUSTRATIVE EXAMPLE

This section takes the reader through the process of planning a simulated mission. The system begins with an initial screen on the Lisp machine, as shown in Figure 4.4, which contains two image panels - the User Prompt Panel and the Select Mission Panel. The former displays the current action to be taken by the human mission planner (the user), while the latter provides a menu of possible AUV missions. At startup, the user is required to respond to a ***select mission*** prompt with a mouse click on the designated mission. The currently available choices are shown in Figure 4.4. At present, only the transit pool mission is fully developed, and this mission is thus used as an example.

After selecting the mission type, the user is presented with a mission specific panel with initially unknown parameters. In this example, as shown in Figure 4.5, a Transit Mission panel is displayed and the user is prompted to enter mission parameter values. Figure 4.5 shows the result of such a selection. In this case, the test pool selected is the NPS swimming pool. An environmental database for this pool, including possible obstacles, is encoded as another KEE unit as shown on Figure 4.2. As can be seen on Figure 4.5, in addition to providing numerical mission parameters, the user must inform the expert system regarding the threat level and also enable or disable hovering mode in the AUV. The reason for the latter choice is that, while hovering mode allows very precise maneuvering, it is very expensive in terms of

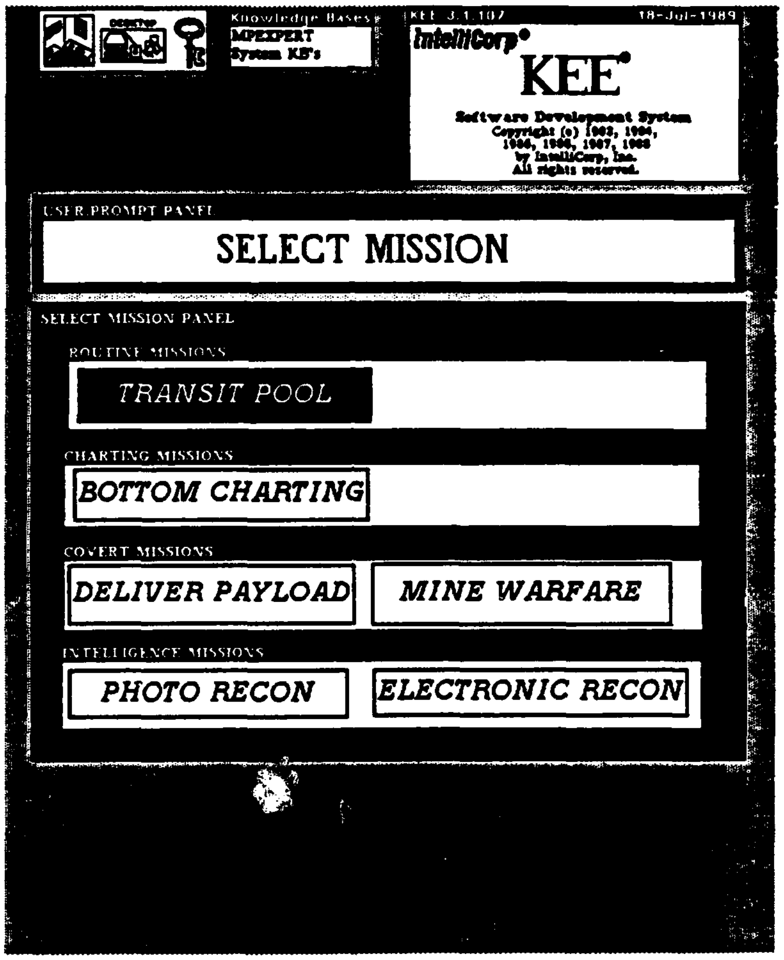
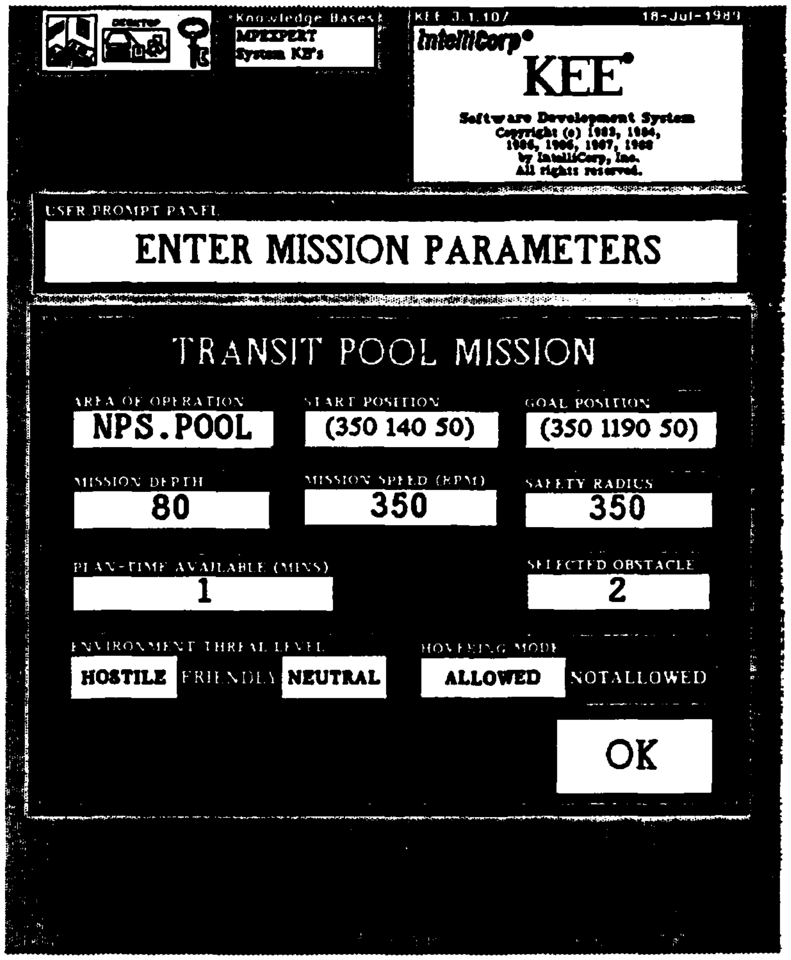


Figure 4.4 Initial Screen on Mission Planning Workstation



ENTER MISSION PARAMETERS

TRANSIT POOL MISSION

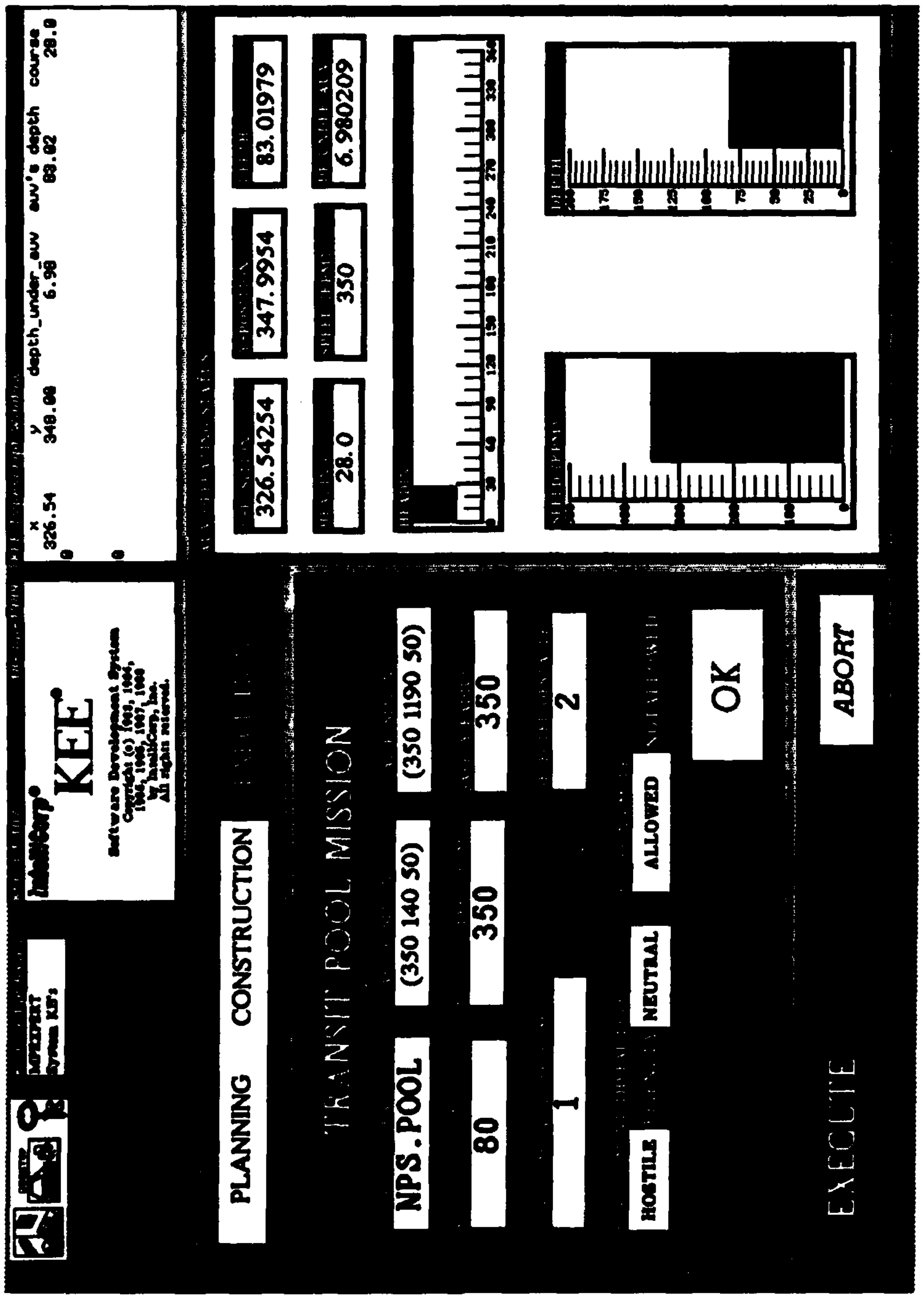
AREA OF OPERATION: NPS.POOL
START POSITION: (350 140 50)
GOAL POSITION: (350 1190 50)
MISSION DEPTH: 80
MISSION SPEED (RPM): 350
SAFETY RADIUS: 350
PLAN-TIME AVAILABLE (MINS): 1
SELECTED OBSTACLE: 2
ENVIRONMENT THREAT LEVEL: HOSTILE FRIENDLY NEUTRAL
HOVERING MODE: ALLOWED NOTALLOWED

Figure 4.5 Screen for Transit Pool Mission

energy and time requirements. In Figure 4.5, the user has indicated a ***friendly*** threat level and hovering ***not allowed***. He has also designated an available planning time of *only one minute*. In making these choices, the user has indicated considerable urgency in getting a mission under way and that rapid, rather than precise transit to the goal is desired. After entering all parameters, the user then mouse clicks on OK to indicate completion.

At this point, the User Prompt Panel at the top is replaced by a Mission Phase Panel (Figure 4.6), which shows the phase of the mission at any given moment. When the mission construction is completed, the system is ready to commence execution phase. During execution phase, two new panels are displayed (Figure 4.6): the AUV Operating Status Panel and the Execute-Abort Panel. The former panel is displayed on the right and it shows the status of the vehicle at any moment, while the latter panel is displayed just below the Transit Pool mission panel and it prompts the user to either proceed with execution or abort the mission. Selecting ***abort*** will abort the mission and bring the system back to the initial screen (Figure 4.4). On the other hand, selecting ***execute*** will initiate a simulated mission on the SGI graphics workstation.

The mission type and parameter selections indicated on Figures 4.4 and 4.5 result in the choice of heuristic search as the only acceptable method of path planning for a mission of this urgency under the specified conditions. The resulting path that



```
x         y        depth_under_auv   auv's depth   course
326.54    349.88   6.98              83.02         28.0
```

326.54254 347.9954 83.01979
28.0 350 6.980209

NPS.POOL (350 140 50) (350 1190 50)
80 350 350
1 2

Figure 4.6 Screen at Start of Mission Execution Phase

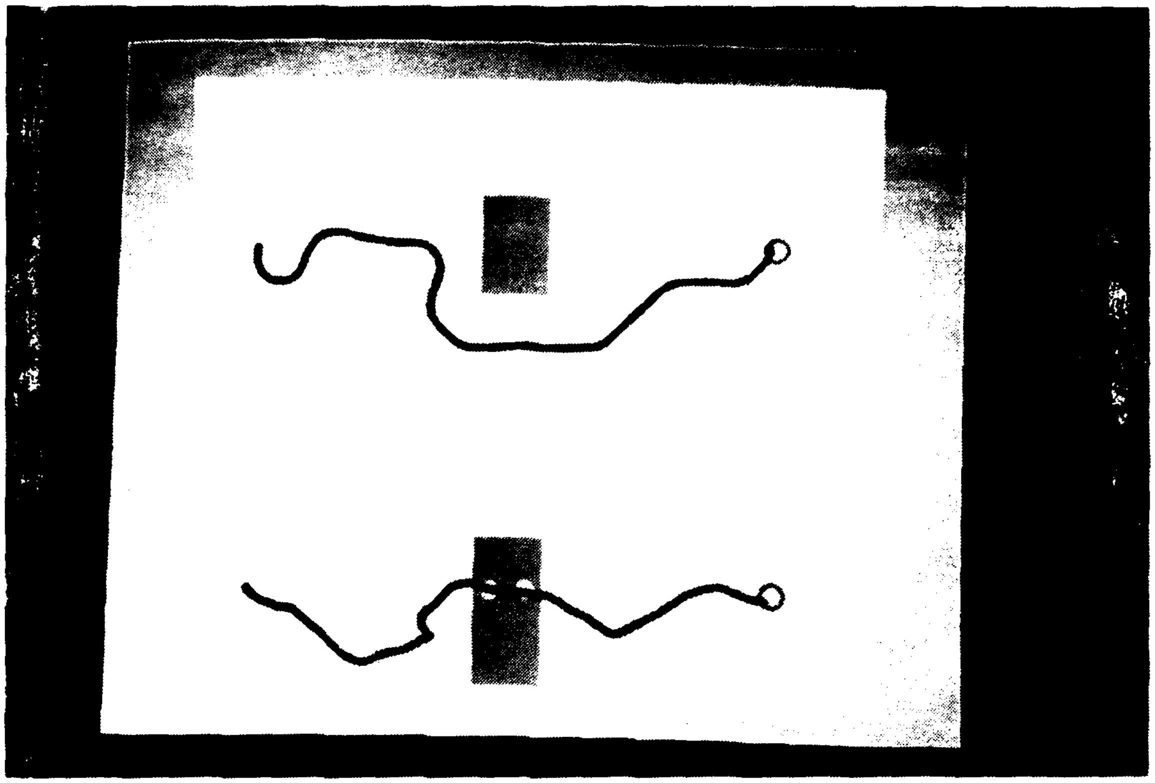


Figure 4.7 Symbolics Color Side Monitor Showing Top View (Upper Image) and Side View (Lower Image) of Waypoints and Vehicle Trajectory for Transit Pool Mission

is displayed on the Symbolics Color Monitor is shown in Figure 4.7; the upper image of the figure shows a top view of the path in the pool environment, while the lower image shows the corresponding side view. The path is represented by a series of dots designating the waypoints. This figure also shows the trajectory followed by the simulated AUV in attempting to transit the specified waypoints.

It should be noted that while the prohibition on the use of thrusters by the human mission planner prevents the AUV from passing through all waypoints, it does successfully reach the specified goal. In accomplishing this task, the navigator shown on Fig. 3.2 used an extremely simple scheme in which desired speed and heading are derived by simply aiming the vehicle at the next waypoint until it enters the proximity of the selected waypoint. The proximity criterion used is a spherical region of radius one grid unit around the waypoint. At that time, the navigator switches to the next waypoint and calculates a new course and speed. Of course, other navigation/guidance methods enable more accurate transiting of waypoints [Ref. 27, 28, 29], but since precise path following is not required in the specified transit mission, the above described simpler approach was used.

It should be observed from Figure 4.7 that the path selected by heuristic search is not optimal; a shorter path results from simply going around the obstacle at the prescribed mission depth. Indeed, the use of A* search would yield this path. However, for the pool size used in this experiment, A* search requires approximately 20 minutes and, as shown on Figure 4.5, in this instance the human mission planner was unwilling to allow this much time for path planning. As a side remark relating to Figure 4.7, one of the features of the heuristic search method used in this research, is that when an obstacle is encountered in a friendly environment, the path planner follows a rising trajectory while trying to go around the obstacle in the hope that a way *over* it can be found without going all the way arounu it. This behavior is clearly evident in the figure.

E. SUMMARY

This chapter presents an in-depth description of the Mission Planning Expert System and its associated Mission Planning and Control Workstation for the NPS AUV. Its structure, as well as the design and development using the object-oriented and rule-based paradigm offered by the KEE expert system shell, is also described in detail. Finally, an example is given that takes the reader through the mission planning phases using the Mission Planning workstation.

V. HEURISTIC SEARCH

A. INTRODUCTION

The Heuristic search method is designed for autonomous vehicles in a cluttered underwater environment. It is an informed search strategy which provides a semi-optimizing solution [Ref. 26] to guiding the vehicle to a specified goal location while maintaining a given transit depth.

As the name suggests, the algorithm is based on heuristics. The specific heuristics used are meant to closely model human behavior in its reasoning decision-making concerning which route to take. These heuristics not only provide local cost optimization decisions but also endow the vehicle with obstacle avoidance and clearance capabilities required for it to operate autonomously.

The dominant characteristics of this method that set it apart from the traditional AI search methods such as A* and Best-first are:

1. It does not require the use of an ***agenda*** [Ref. 9, 10] of unexplored paths.
2. It makes extensive use of heuristics for path-search as well as for obstacle clearance.
3. It does not require complete a priori information on the environment.
4. It is capable of dealing with uncharted obstacles.
5. It is relatively much faster.
6. It can be extended to deal with dynamic obstacles.

B. PRELIMINARY DEFINITIONS AND NOTATIONS

In order to discuss the Heuristic search precisely, it is first necessary to define the terms as well as the notations used throughout this chapter. The definitions of the terms and notations are tabulated in Tables 5.1 and Table 5.2 respectively. In addition, throughout this chapter, the term obstacle is used to refer to virtual obstacles (see Section E of Chapter III).

TABLE 5.1 DEFINITION OF TERMS

TERMINOLOGY	DEFINITION
Goal	The Goal position or destination.
Start	The start position.
state	The tuple (vehicle-heading, position).
candidate successor	One of the possible successors of the current states.
candidate position	The position coordinates of the candidate successor.
mission-depth	The depth specified for the current mission.

TABLE 5.2 NOTATIONS

NOTATIONS	DEFINITION
P_{start}	Start position
P_{goal}	Goal position
$P(x,y,z)$	Position located at coordinate (x,y,z)
P_n	Current vehicle position
$P_n(x,y,z)$	Current vehicle position at coordinate (x,y,z)
P_k	Vehicle position after its kth move from P_{start}
S_n	Current vehicle state
$S_n(theta,P_n)$	Current vehicle state with a heading of theta at P_n
CS_{n+1}	One of the candidate successor states
CP_{n+1}	The position corresponding to candidate successor state CS_{n+1}
S_{goal}	The vehicle state at the Goal
Horiz_Dist(A,B)	Horizontal distance between positions A and B
Depth_Change(A,B)	Vertical distance between positions A and B
EF	Evaluation Function
$EF(CS_{n+1})$	Evaluation Function of candidate successor CS_{n+1}
EC	Estimated Cost Function
$EC(CS_{n+1})$	Estimated Cost Function of candidate successor CS_{n+1}
LC	Local Cost Function
LC(A,B)	Local cost incurred in moving from point A to B
TC(A,B)	Translational cost incurred in moving from pt. A to B
RC(A,B)	Rotational cost incurred in moving from point A to B
$PM(CS_{n+1})$	Path-marking value of candidate successor CS_{n+1}

C. SUCCESSOR POSITIONS

In a 3-dimension underwater environment, a point or position on the grid has 26 possible candidate successors. Figure 5.1 shows the names adopted for these candidates; the successors of a position are named according to their directions with respect to that position. The top successors are prefixed with a 't', while the bottom successors have a 'b' prefix. In pruning the search tree, however, only viable candidates of a given state, are searched. These potentially viable successors form groups called ***successor sets***. Table 5.3 shows the different successor sets defined; note that the sets are *not* disjoint.

The successor set currently selected for search is referred to as the ***active successor set***, and its members are called the ***candidate successors***. Which successor set is active in a given situation depends on the ***search mode*** (see Section H of this chapter) in force. Moreover, more than one successor set may be active in a given situation; in this case the union of these sets is the active set. A candidate successor is said to be "**open**" if it is not an obstacle, and "**closed**" if it is an obstacle. Similarly, an active successor set may be "completely open", "partially open", or "completely closed". It is "**completely open**" if all the candidate successors in the set are not obstacles. It is "**partially open**" if there is at least one successor within the set that is *not* an obstacle. Finally, the active successor set is "**completely closed**" if all the candidate successors in the set are obstacles.

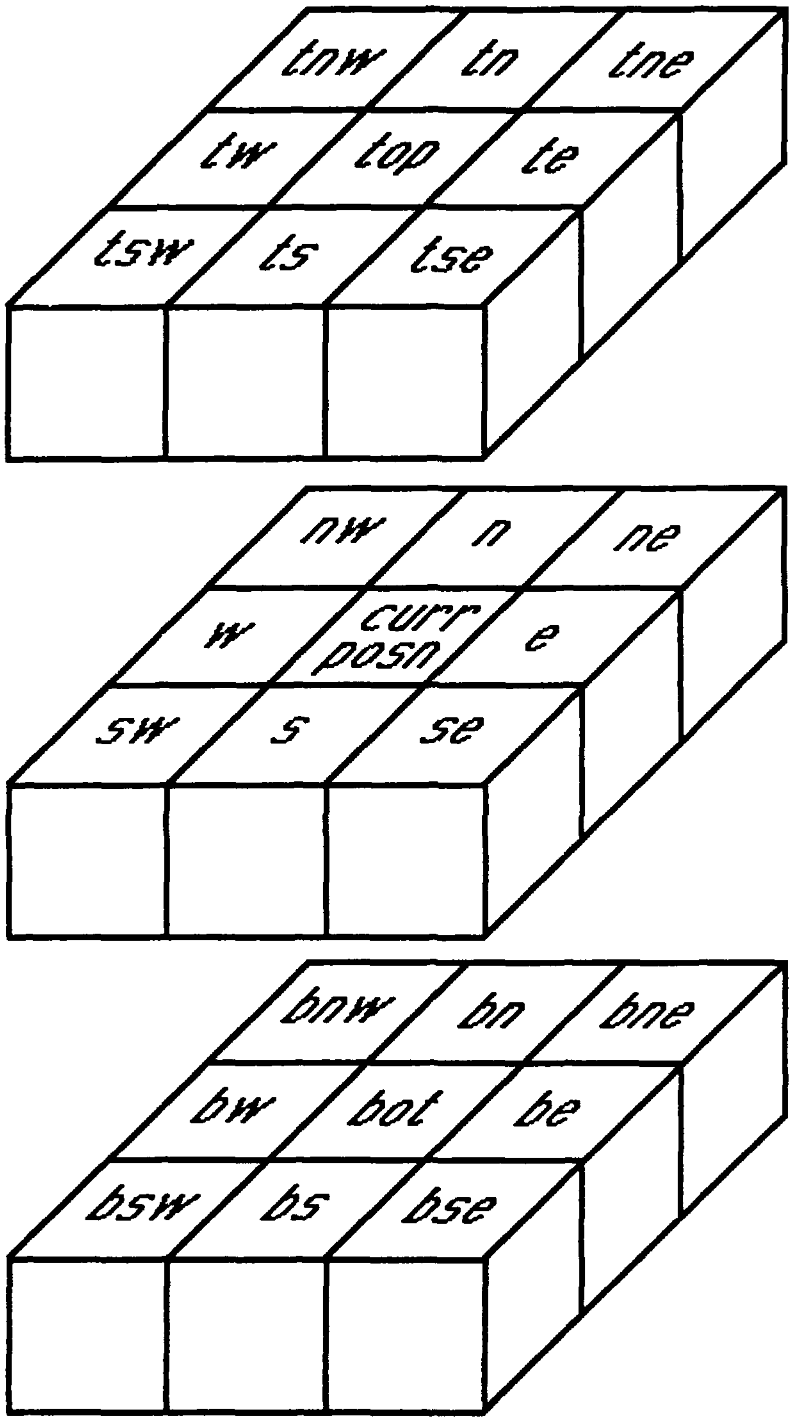


Figure 5.1 3D Candidate Successors of a State

TABLE 5.3 SUCCESSOR SETS OF THE CURRENT STATE S_n(theta,P_n(x,y,z))

SUCCESSOR SET	DESCRIPTION
fwd-level	The 3 successors in the forward direction with respect to the vehicle heading, theta, and having the same depth z, as the current position.
fwd-rise	The 3 successors in the forward direction with respect to the vehicle heading, theta, at a depth of z-1.
fwd-dive	The 3 successors in the forward direction with respect to the vehicle heading, theta, at a depth of z+1.
top-fwd-rl	The 2 successors in the forward direction, one on the right and the other on the left with respect to the vehicle heading, theta, and having a depth of z-1.
bot-fwd-rl	The counterpart of the top-fwd-rl successor set except that the 2 successors are at a depth of z+1.
top-rl	The 2 successors in the same vertical plane as the vehicle, one on the right and the other on the left with respect to the vehicle heading, theta, and having a depth of z-1.
bot-rl	The counterpart of top-rl successor except the two successors are at a depth of z+1.
fwd-top	The single successor in the forward direction with respect to the vehicle heading, and having a depth z-1.
fwd-bot	The single successor in the forward direction with respect to the vehicle heading and having a depth of z+1.
right-left	The 2 successors on the right and left sides of the vehicle.
back-up	The 9 successors behind the vehicle.
top	The single successor directly above the vehicle.
bottom	The single successor directly below the vehicle.

It should also be noted that the successor set of a state is a function of the heading of that state. For instance, the forward-level successors for a current heading of 0 degrees is the set [ne, n, nw], while the same for a heading of 90 degrees is [ne, e, se]. With this dependence on heading, it is more accurate to speak of the successors of a state rather than the successors of a position. To illustrate the different successor sets, Figure 5.2 shows an example of the different successor sets for a vehicle heading of 270 degrees. In addition, the term ***forward position*** is used to refer to any successor set in front of the vehicle. In the example of Figure 5.2, the forward position refers to one or more of the following sets: the fwd-level, fwd-rise, and fwd-dive successor sets.

D. HEURISTICS

The heuristics employed offer advice on which set of successors of a state to try for further search under a given circumstance. Two classes of heuristics are defined - ***General heuristics*** and ***Obstacle Clearance heuristics***. General heuristics are applicable under all circumstances or modes, providing guidance on the choice of the best successor position while Obstacle Clearance heuristics suggests a systematic approach to searching for "a way out" when an obstacle is encountered.

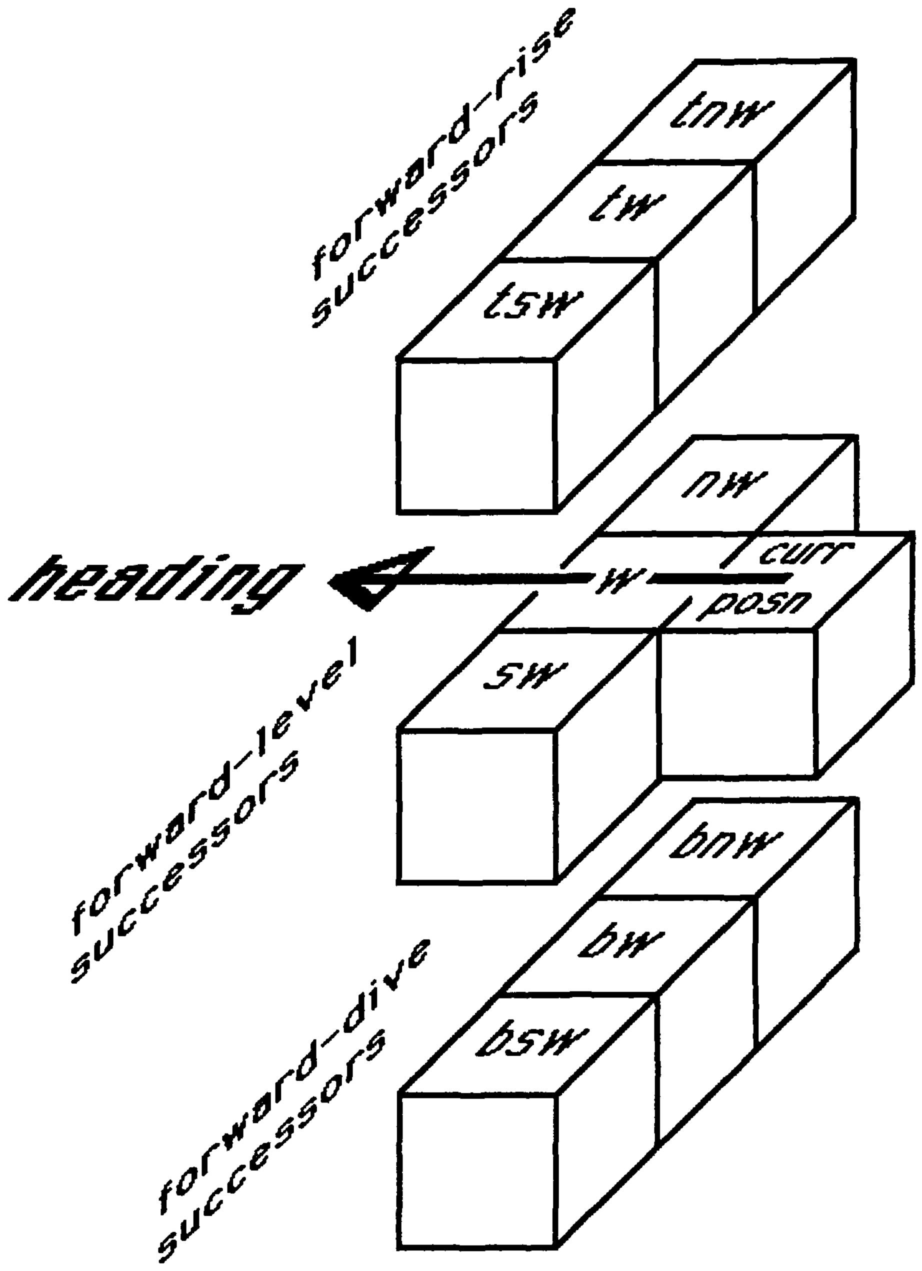


Figure 5.2 Forward Successor Sets for a Heading of 270°

The following general heuristics are adopted:

1. Move toward the goal whenever possible.
2. Prefer to move in the direction of current heading
3. Try not to visit the positions already explored.
4. Keep to the specified mission-depth as far as possible.
5. Search forward successor positions as far as possible.

Obstacle clearance heuristics used are as follows:

6. Prefer to search bottom successors (bottom-search) or top successors (top-search) as determined by the rule-based system.
7. In either case, prefer to move along the diagonal of the obstacle until it is cleared.

E. ENERGY COST MEASURE

All paths have an associated cost in terms of the amount of energy expended in traversing it. The path-planning problem requires finding a reasonable cost (semi-optimizing) path between the start and the goal positions. Thus, some measure of energy cost has to be adopted. In this study, the energy cost is normalized to distance units, which is *inches* (the unit used to measure the size of the vehicle); this unit shall be referred to as the ***normalized energy unit***, or simply the ***energy unit***. For example, suppose the distance between point A and point B on a horizontal plane is 100 inches then energy expended in moving from A to B is 100 energy units.

F. EVALUATION FUNCTION (EF)

Heuristics 1, 2 and 3 require an Evaluation Function to estimate **the cost of moving from a given state to the Goal**. At each state, this cost is evaluated for all candidate successors and the one with the *lowest* evaluation function is chosen as the best successor of the current state. Note that the evaluation function does not include the *accumulated* cost of moving the vehicle from the Start to its current state. Thus, unlike A* search which performs "global optimization", heuristic search is guided by local optimization.

The Evaluation Function has 2 main components:

1. Local Cost (LC) of moving from the current state to a candidate successor state.
2. Estimated Cost (EC) of moving from the candidate successor state to the Goal position.

Mathematically, this can be expressed as follows:

$$EF(CS_{n+1}) = LC(S_n, CS_{n+1}) + EC(CS_{n+1}) \tag{5.1}$$

1. LOCAL COST FUNCTION (LC)

The Local Cost function computes the energy required to move the vehicle from its current state (S_n) to a candidate successor state (CS_{n+1}). It is the sum of two components:

1. Translational Cost to move the vehicle from point P_n to CP_{n+1}.
2. Rotational Cost required to change the heading of the vehicle in moving from S_n to CS_{n+1}.

Mathematically, it is expressed as follows:

$$LC(S_n, CS_{n+1}) = TC(P_n, CP_{n+1}) + RC(S_n, CS_{n+1}) \quad (5.2)$$

a. Translational Cost (TC)

The translational cost is different for horizontal and vertical maneuvers. Here, it is assumed that the cost rate (i.e., the energy expended per unit distance) for vertical movement (depth changes) is greater than that for horizontal movement by a factor of 1.2. For example, suppose the Euclidean distance between point A and point B is 100 inches; if A and B lie on the same horizontal plane, then energy expended is 100 inches, whereas, if A and B were in the same vertical plane, the cost would be 120 inches. Formally, the Translational Cost in moving from point A to point B is defined as follows:

$$TC(A,B) = Horiz\_Dist(A,B) + 1.2 * Depth\_Change(A,B) \quad (5.3a)$$

Thus, in moving from current state Sn to a candidate successor CSn+1, the translational cost incurred is given by:

$$TC(P_n, CP_{n+1}) = Horiz\_Dist(P_n, CP_{n+1}) + 1.2 * Depth\_Change(P_n, CP_{n+1}) \quad (5.3b)$$

b. *Rotational Cost (RC)*

Due to the inertia of a vehicle, there is a cost associated in changing the heading of a vehicle, and this is accounted for by the concept of rotational cost. This cost tends to make a vehicle maintain its current direction of movement. Formally, the rotational cost is defined as the amount of energy expended (in normalized energy units) in changing the vehicle heading while moving from S_n to CS_{n+1}. In general, rotational cost varies with the turning angle; the larger the turning angle, the larger the cost. Table 5.4 below shows the rotational cost variation with angle. Note that a 45 degree turning angle means either a 45 degree left turn or a 45 degree right turn with respect to the current heading of the vehicle.

TABLE 5.4 ROTATIONAL COST (in normalized energy units)

Turning Angle (degrees)	0	45	90	135	180
Rotational Cost	0	7	35	70	140

2. ESTIMATED COST FUNCTION (EC)

The Estimated Cost represents the ***minimum*** estimated energy required by the vehicle in moving from a candidate successor state, CS_{n+1}, to the Goal. It is the minimum cost that will be incurred if that candidate is chosen, regardless of the remaining path chosen from it to the Goal. Since a lower Estimated Cost results in a correspondingly lower Evaluation Function, a candidate successor with a lower cost estimate is favored. Thus, the Estimated Cost Function serves as a "pulling force", drawing the vehicle towards the Goal. It has two components - the minimum expected translation cost and the minimum expected rotational cost. Mathematically, the total cost is expressed as:

$$EC(CS_{n+1}) = TC(CP_{n+1},P_{goal}) + RC(CS_{n+1},S_{goal}) + PM(CS_{n+1}) \tag{5.4}$$

a. Minimum Expected Translational Cost

This component, $TC(CP_{n+1},P_{goal})$, decreases with distance from the Goal. Thus, candidate successor positions nearer the Goal are favored, thereby aiding the vehicle to move towards the Goal. This quantity is computed in the same manner as the translational cost component of the Local Cost function, with the appropriate parameter substitutions.

b. Minimum Expected Rotational Cost

Figure 5.3 illustrates the concept of minimum expected rotational cost, $RC(CS_{n+1},S_{goal})$. It is the minimum turning cost that will be incurred in moving the vehicle from the candidate successor to the Goal. Like its translational counterpart, its role is to enhance the vehicle's tendency to move toward the Goal by favoring successors with a lower minimum. This quantity is computed using the same table (Table 5.4).

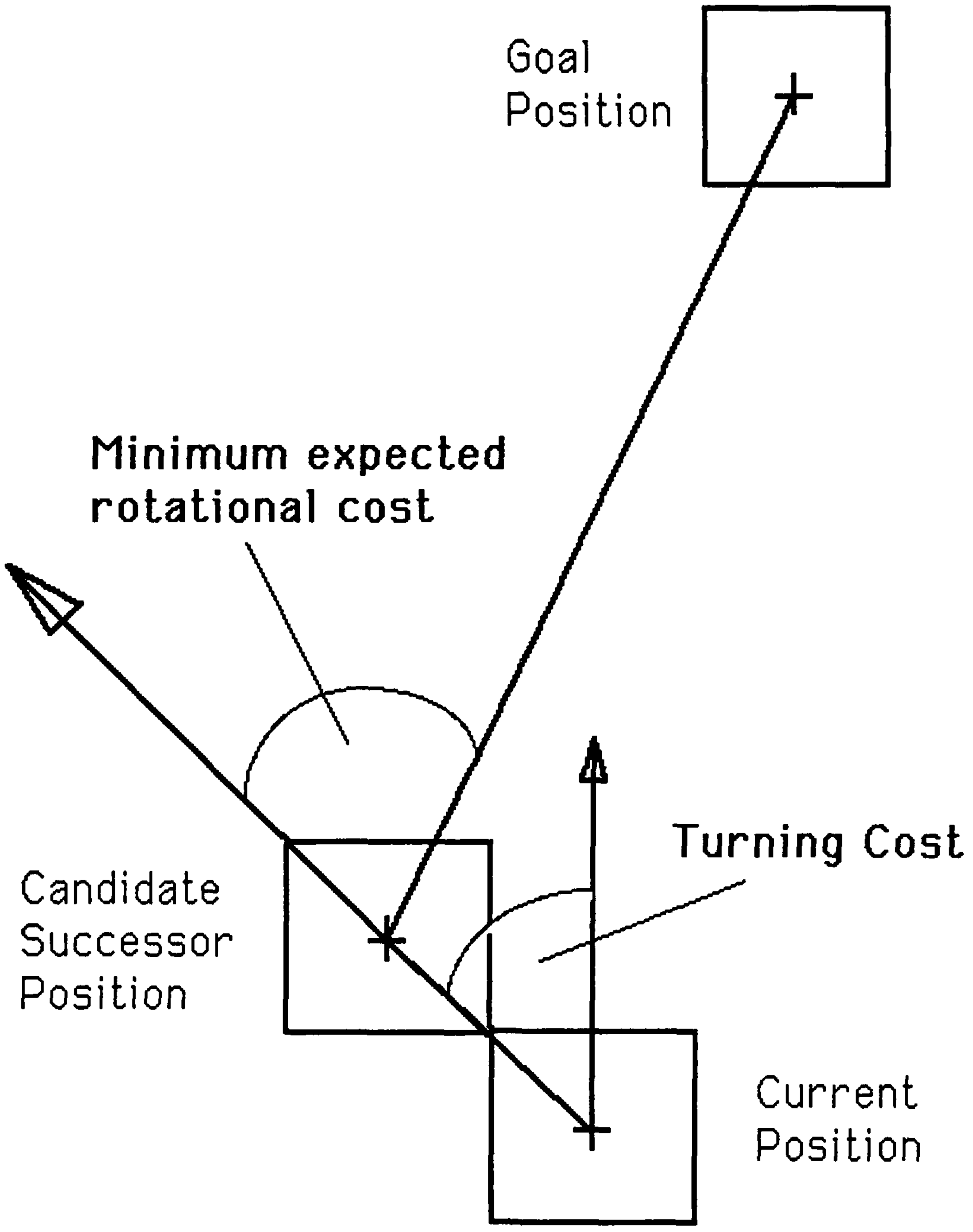

Goal Position
Minimum expected rotational cost
Turning Cost
Candidate Successor Position
Current Position

Figure 5.3 Minimum Expected Rotational Cost

c. Path Marking

Path marking [Ref. 8] is a concept introduced to implement the Heuristic 3, which says "to prefer candidate successor whose positions are not already explored". It provides a means for the vehicle to "memorize" the positions already visited. The technique works as follows. The path marking value of each position is initially zero. Whenever the vehicle moves from state S_n to CS_{n+1}, a path marking value, $PM(CS_{n+1})$, equivalent to the Local Cost $LC(S_n,CS_{n+1})$, is assigned to the position P_n (corresponding to state S_n). This value serves to increase the Evaluation Function of position P_n when it is next evaluated as a candidate successor, thereby reducing its favorability and its chances of being chosen as the best successor. In this manner, Heuristic 3 is facilitated.

To summarize the various component costs discussed in this section (Section F), Figure 5.4 shows the entire cost structure.

G. OBSTACLE CLEARANCE

The obstacle clearance heuristics exploit the fact that obstacles in the real world are largely high or wide. Odd-shaped obstacles can be approximated by these two shapes. As shown in Figure 5.5, for high obstacles, the shortest path is to move horizontally *around* the obstacle, whereas for wide obstacles, the shortest path is to move vertically *over* or *under* the obstacles. However, without any knowledge of the

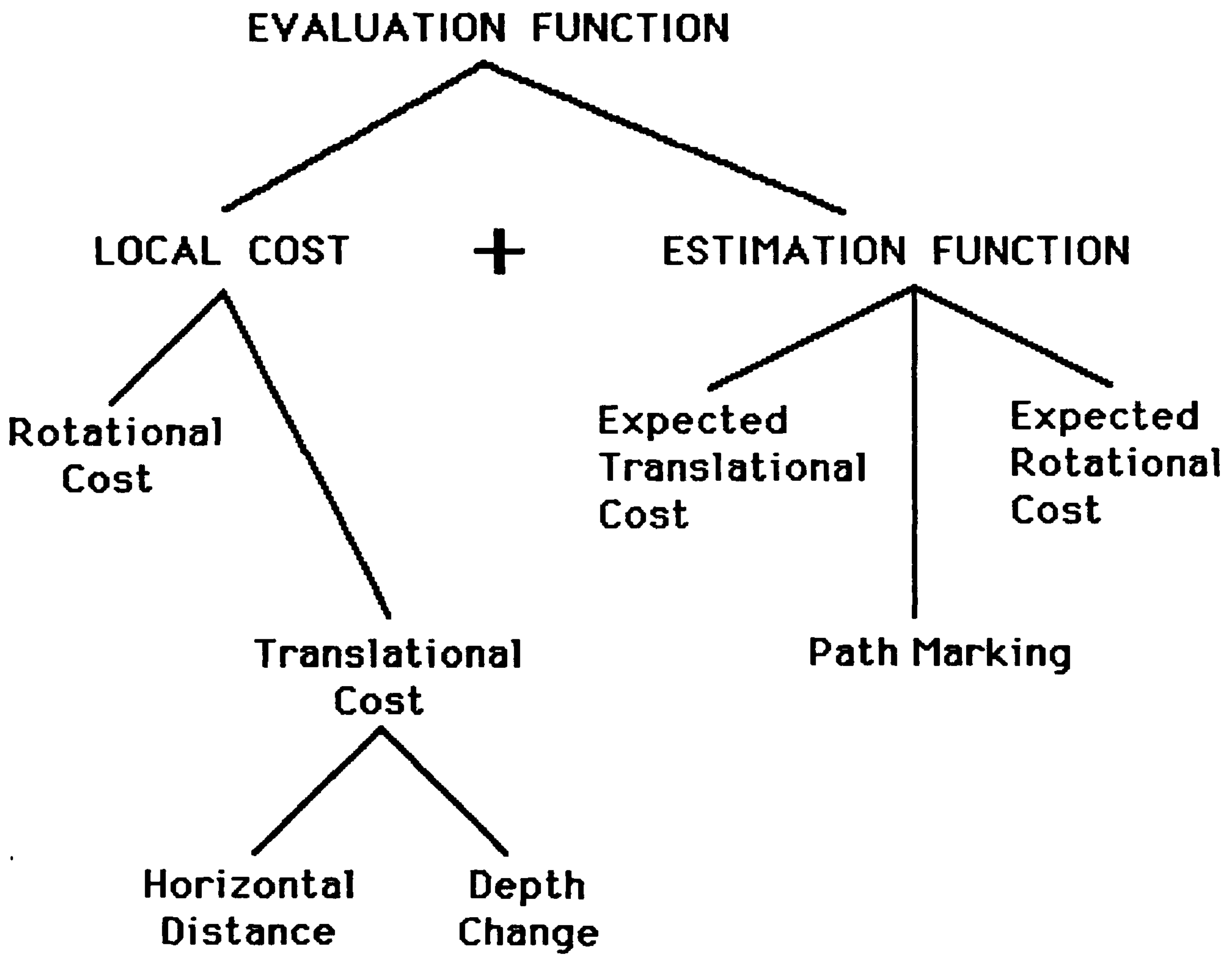


Figure 5.4 Cost Structure

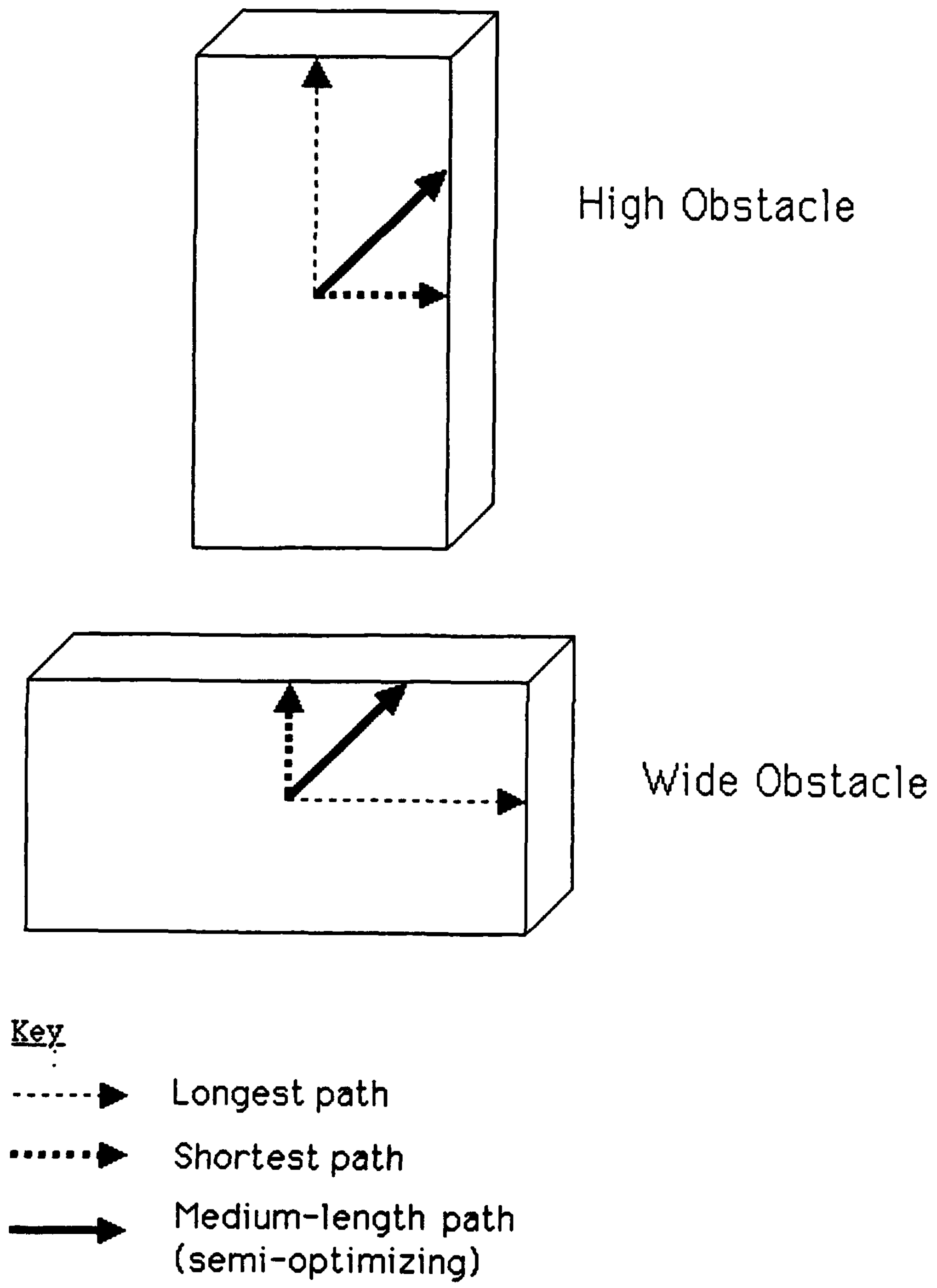


Figure 5.5 Semi-Optimizing Paths Around Obstacles

disposition of the obstacle, it is not possible to determine which way is shortest. A compromise, semi-optimizing solution is to move along the diagonal of the obstacle, as illustrated in the figure. Thus, in the absence of any a priori information concerning the shape and size of the obstacle encountered, a reasonable strategy would be to move diagonally along the obstacle wall whenever possible. This heuristic is realized by defining a preferred ***successor search sequence*** that constrains the vehicle to do just that, as explained in the following sections.

H. MODES OF OPERATION

1. OVERVIEW

The heuristic search algorithm defines three modes of operation: NORMAL mode, OBSTACLE mode, and OBSTACLE-EDGE mode. These modes determine the heuristics that are called into play. Since the latter also affects the successor sets to be searched, the modes are also referred to as search modes. Before proceeding further, it is emphasized again that the term obstacle used in this section refers to virtual obstacles.

Figure 5.6 shows the mode transition flowchart at a high conceptual level. The search process begins in the NORMAL mode where only the general heuristics are employed. It remains in this mode until the vehicle encounters an obstacle blocking its path. It then changes to OBSTACLE mode and calls upon the obstacle

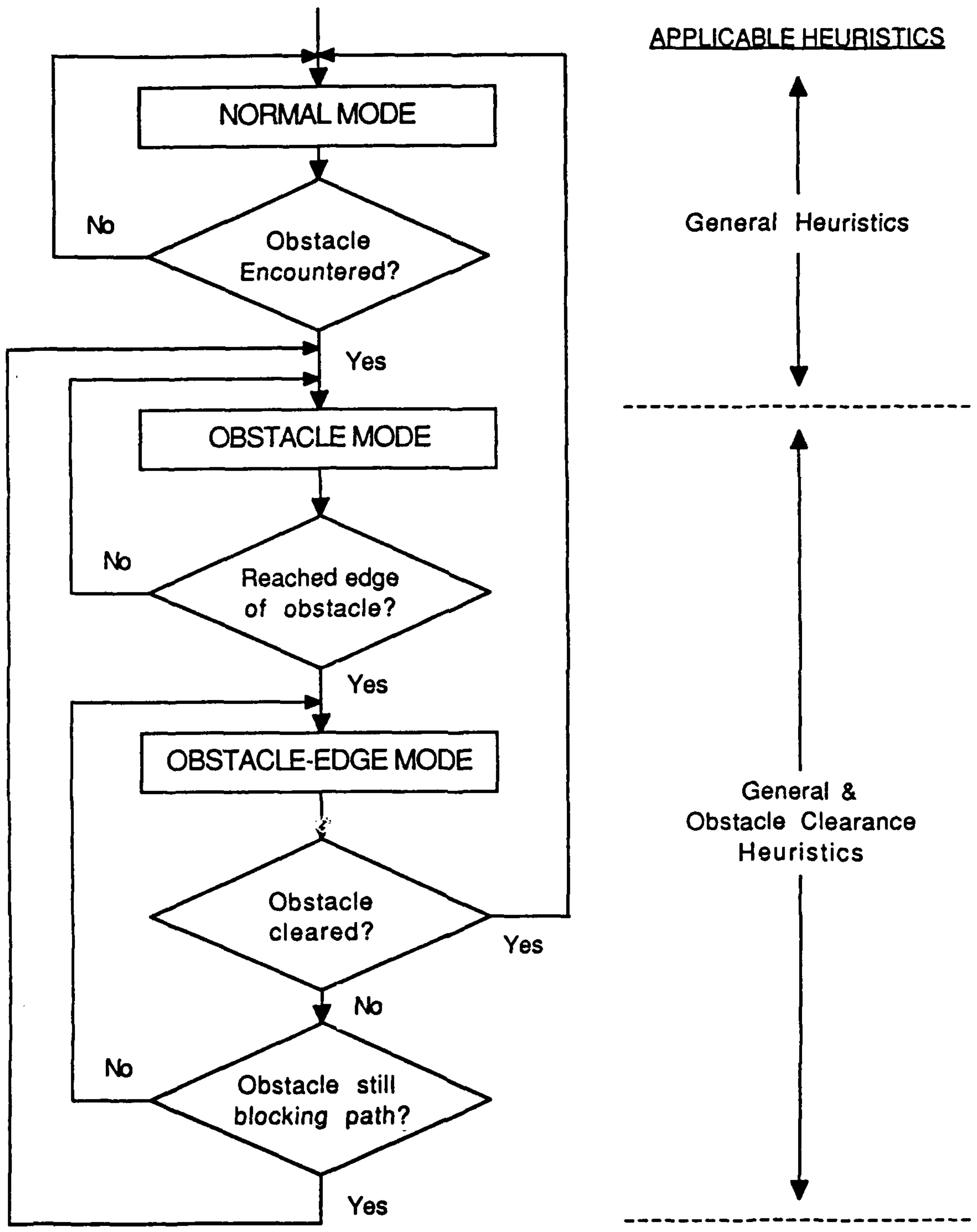


Figure 5.6 Modes of Operation - Conceptual Flowchart

clearance heuristics to guide it. Whenever it reaches an edge of the obstacle, it progresses to OBSTACLE-EDGE mode. The latter mode is required to confirm that the obstacle has indeed been cleared. The criteria for this decision is explained shortly. If confirmation is negative, it returns to OBSTACLE mode; otherwise it switches back to NORMAL mode. Figure 5.7 shows the corresponding flowchart with the actual criteria used to determine the mode transitions; note that the right side of the figure shows the active successor set corresponding to the questions at each stage, in the chart.

2. NORMAL MODE

In NORMAL mode, only the forward positions of the current state are searched. The forward positions may be any one of the three successor sets, namely, the fwd-level, the fwd-rise, and the fwd-dive (see Figure 5.2), depending on the current vehicle depth with respect to the mission-depth. The fwd-level successor set is searched (i.e. is active) when the vehicle is at mission-depth, the fwd-rise when its depth is greater than mission depth, and lastly, the fwd-dive when its depth is less than the required mission-depth. The system enters OBSTACLE mode if and only if the currently active successor set is "completely closed", indicating that an object is blocking its path. Note that in the situation where the active set is "partially open", the vehicle is not considered to have "encountered" an obstacle (it merely came close to one); it therefore remains in NORMAL mode.

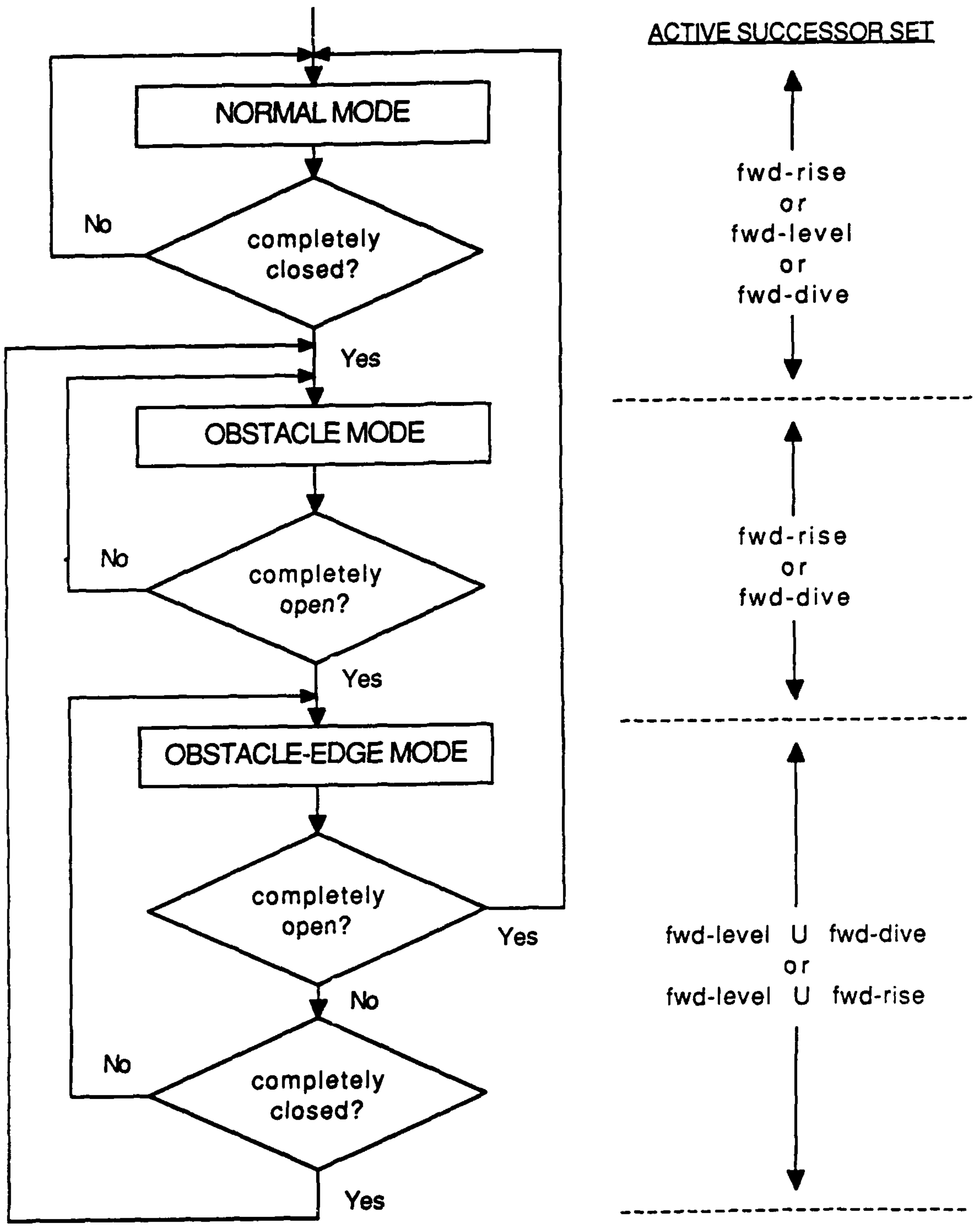


Figure 5.7 Criteria for Mode Transitions

3. OBSTACLE MODE

When in this mode, the vehicle has sensed an obstacle in its path and immediately consults the obstacle clearance heuristics for guidance. Here, the active successor set to be searched is controlled based on a preferred (or prioritized) sequence list. This list defines the order of the successor sets to be examined in turn until an "open" successor is found. Such prioritized search is necessary in order to force the outcome of the search to preferred successor(s) wherever possible. Recall that according to the obstacle clearance heuristics, it is preferable to move forward, and along the diagonal of the obstacle.

To realize the obstacle clearance heuristics, two search sequences are defined, namely, the top-preferred-sequence, and the bottom-preferred-sequence. Which sequence is used depends on whether bottom search or top search is preferred, as determined by the rule-based planner. When the threat level is hostile, the bottom-preferred-sequence is chosen; otherwise, the default top-preferred-sequence is used. The two sequences are defined as follows:

Top-preferred-sequence:

[fwd-rise fwd-level top-rl fwd-top top
fwd-dive bot-rl fwd-bot right-left backup]

Bottom-preferred-sequence:

[fwd-dive fwd-level bot-rl fwd-bot bottom
fwd-rise top-rl fwd-top right-left backup]

The search always begins with the *first* successor set in the sequence chosen. This is the fwd-rise successor set if the top-preferred sequence is selected, and it is the fwd-dive set if the bottom-preferred sequence is chosen. This first successor set is used as the criteria for progressing from OBSTACLE mode to OBSTACLE-EDGE mode. There are three possible cases: the first successor set is "completely open", "partially open" or "completely closed".

If the first successor set is "completely open", it is an indication that the search process has reached an edge of the obstacle, where clearance is possible; in this case, the system progresses to OBSTACLE-EDGE mode. In fact, this is the only situation where the system is allowed to move on to OBSTACLE-EDGE mode. It must be stressed again that this criterion for transiting to OBSTACLE-EDGE mode from OBSTACLE mode applies *only* to the first successor set in the sequence, as shown on the right column of Figure 5.7.

If the first successor set is "partially open", then the best candidate successor is selected, but the system remains in OBSTACLE mode. Lastly, if it is "completely closed", the next successor set in the sequence becomes active and is tried. If this set is also "completely closed", then the next one in line is tried. This continues until an open candidate successor is found. Like the second case, the system remains in OBSTACLE mode. It must be stressed that, if the system remains in OBSTACLE mode, the search in the next cycle will begin again with the first successor set.

4. OBSTACLE-EDGE MODE

This is a transitory mode which serves to confirm that the obstacle has indeed been cleared. This mode is necessary to prevent the system from going to NORMAL mode prematurely, and causing it to return to OBSTACLE mode immediately because the obstacle is not fully cleared.

When the system enters OBSTACLE-EDGE mode, the vehicle may have deviated away from the mission depth, considering that it was previously in OBSTACLE mode, trying to find a way out. Thus, the active successor sets chosen in OBSTACLE-EDGE mode should attempt to bring the vehicle back to the mission depth. To achieve this, there are two alternative active successor sets that can be searched: the first set is the union of fwd-level and fwd-rise successor sets (with no priority between members of the union), and the second set is the union of fwd-level and fwd-dive successor sets. Each of these contain six candidate successors. The first set is used when the current vehicle depth is greater than the mission depth; otherwise the second set is used. The reason for including the fwd-level successor set in the two alternatives is to allow the vehicle to move forward horizontally whenever moving towards the mission depth is not possible.

After the active set (containing six candidates) has been chosen as described, there are again the usual three possibilities: the set is "completely closed", "partially open", or "completely closed". If it is "completely closed", then a wall of obstacles is on its path, and it regresses to OBSTACLE mode. Otherwise, if the active set is "completely open", there is a high chance that the obstacle has been cleared, and it proceeds to NORMAL mode. Lastly, if the successors are "partially open", then the obstacle is still in its immediate vicinity; in this case, it chooses the best successor (according to the general heuristics), but remains in OBSTACLE-EDGE mode.

I. AN ILLUSTRATIVE EXAMPLE

To illustrate the mode transitions during obstacle clearance, consider the example shown in Figures 5.8a and 5.8b. Figure 5.8b shows the corresponding front, side and top views of Figure 5.8a. Note that virtual obstacles are shown in the diagrams.

In the situation depicted, the Goal is assumed to be far away on the other side of the wall (Figure 5.8a), and near the X=0 plane. It is also assumed that the system is in NORMAL mode when the vehicle is at position P_1 = P(5,2,4) -- at the mission depth, z = 4 -- and heading in the direction of the Y-axis. In this state, the active successor set is the fwd-level set { P(4,3,4), P(5,3,4), P(6,3,4) }. Since this set of coordinates are all obstacles, it is "completely closed"; thus, the system changes its mode to OBSTACLE mode, while still at position P_1.

Assuming the mission planner decides that top search is preferred, the top-preferred-sequence is used, and the first successor set in this sequence is fwd-rise. Since this set is also "completely closed", the next set in the sequence, namely the

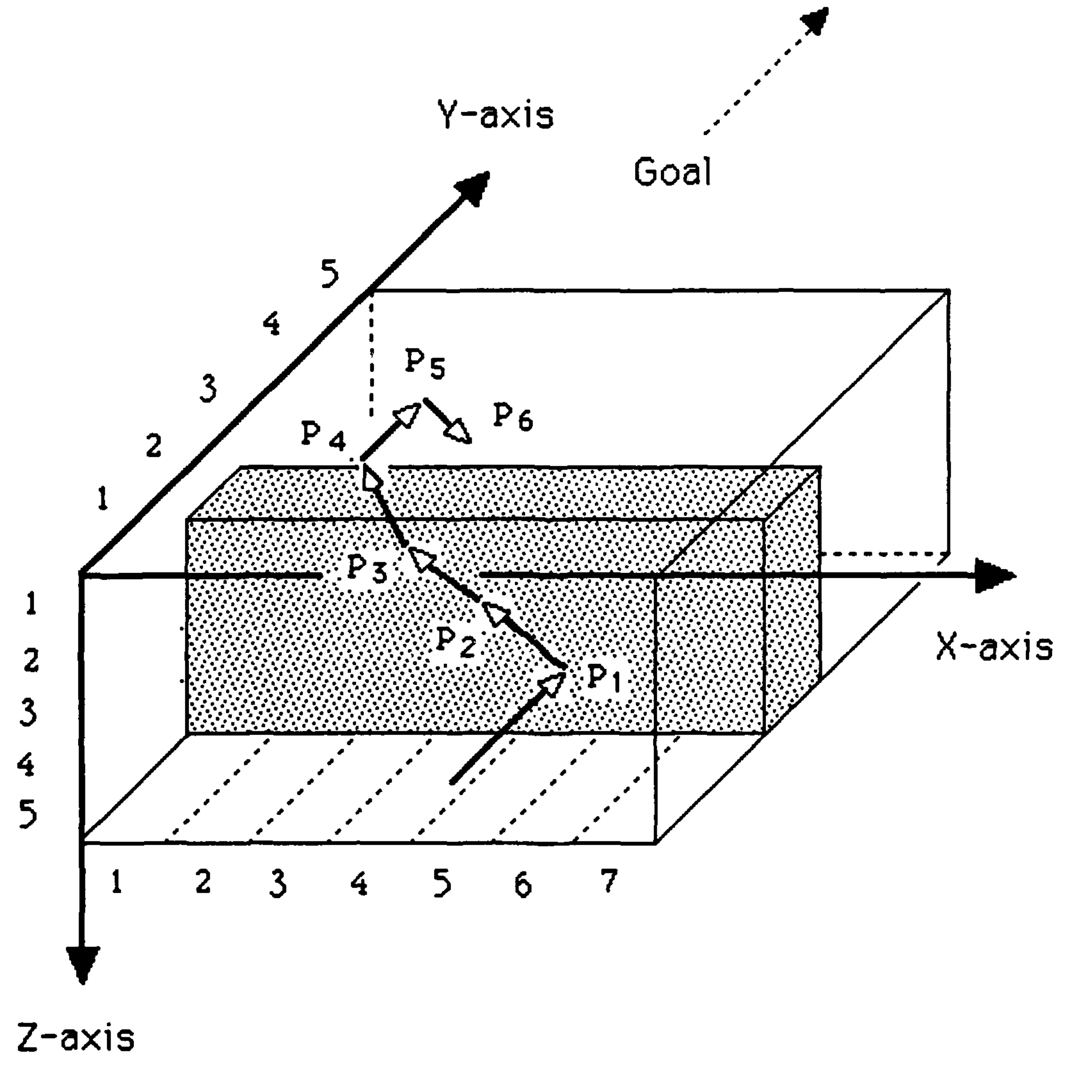


Figure 5.8a An Example of Obstacle Clearance

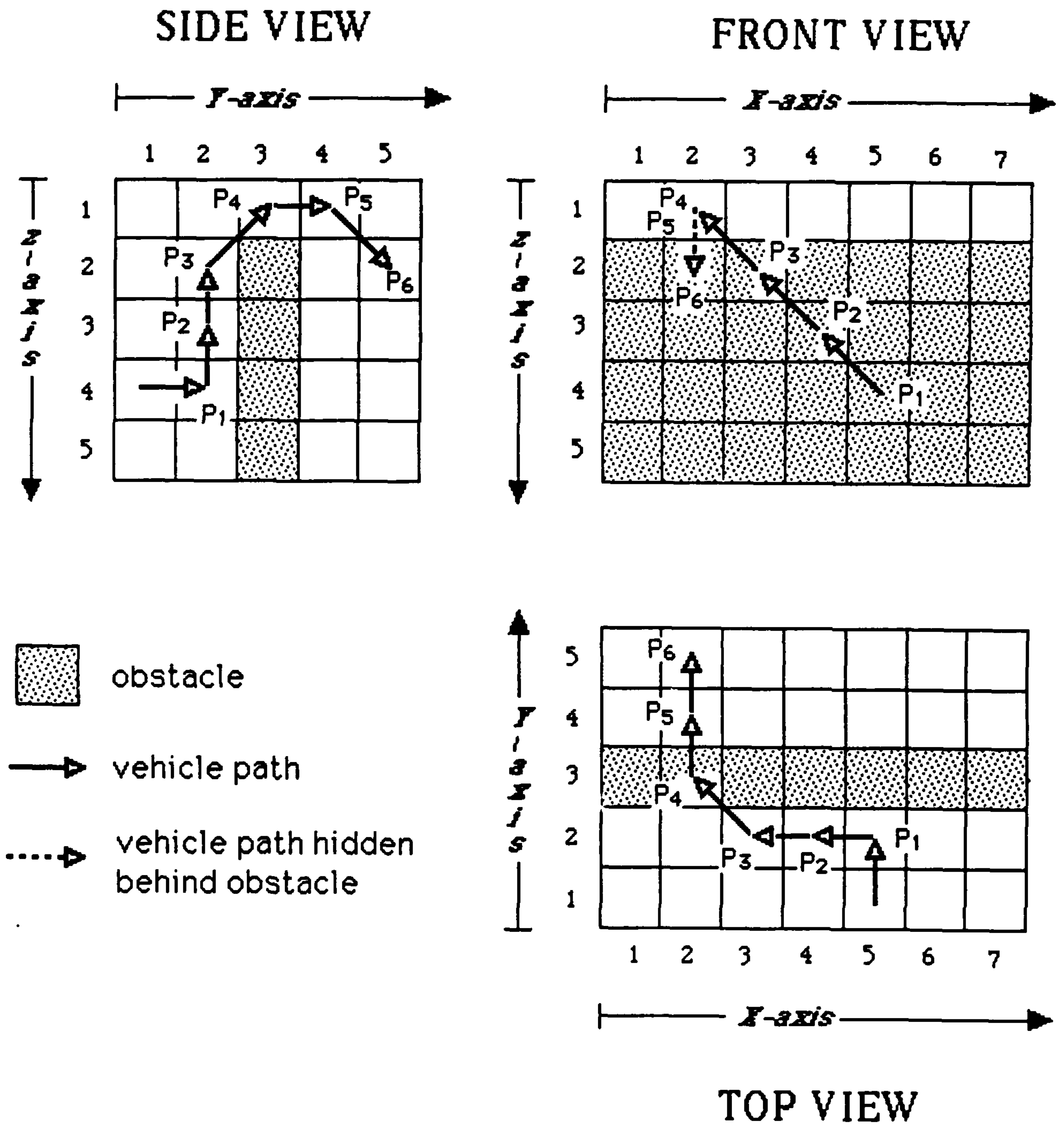


Figure 5.8b An Example of Obstacle Clearance

fwd-level is tried. As noted earlier, this set is "completely closed"; so, the next successor set in the sequence, namely, the top-rl set is searched. This set contains two candidate successors, P(4,2,3) and P(6,2,3), both of which are "open". Suppose that P(4,2,3) is selected (by the general heuristics) as the successor; so the best successor is P_2 = P(4,2,3), but the system remains in OBSTACLE mode (since the first successor set is not "completely open").

Note that the vehicle heading at position P_2 has changed to the negative X-axis direction. With the system still in OBSTACLE mode, the search begins again with the first successor set of the top-preferred-sequence. At this state, the fwd-rise successor set is { P(3,1,2), P(3,2,2), P(3,3,2) }. Of these candidate successors, P(3,2,2) is chosen as the best successor, since P(3,3,2) is an obstacle and P(3,1,2) has a high expected rotational cost. Thus, the best successor is P_3 = P(3,2,2). Here again, the system stays in OBSTACLE mode because the first successor set is not "completely open".

At position P_3, the search commences with the fwd-rise set which is the set { P(2,1,1), P(2,2,1), P(2,3,1) }. Of the three candidates, P(2,3,1) is chosen because it is nearest to the goal and also, because it has the lowest expected rotational cost. Hence, P_4 = P(2,3,1). This time, the situation has improved; since this first successor set in the sequence is "completely open", the system can progress to the OBSTACLE-EDGE mode.

Note that in traversing from P_1, through P_2 and P_3, and then to P_4, the vehicle is actually constrained to move diagonally along the obstacle, in accord with the obstacle clearance heuristics. Moreover, at P_4, the vehicle has actually crossed the edge of the obstacle; hence, the term OBSTACLE-EDGE mode.

When the vehicle reaches position P_4, its depth is less than the mission depth (z=4); so, the active successor set during OBSTACLE-EDGE mode is

fwd-level U fwd-dive

= { P(1,4,1), P(2,4,1), P(3,4,1) } U { P(1,4,2), P(2,4,2), P(3,4,2) }

= { P(1,4,1), P(2,4,1), P(3,4,1), P(1,4,2), P(2,4,2), P(3,4,2) }

Among these, P_5 = P(2,4,1) is chosen as the best successor, because the other candidates would have incurred greater cost by changing either the vehicle depth or its heading. Further, since this set is "completely open", the system proceeds to NORMAL mode, signalling that the obstacle has been cleared. From then on, the NORMAL mode heuristics would constrain the vehicle to move down towards the mission depth, by continuing to search only the fwd-dive successor set until it reaches it.

J. SUMMARY

This chapter discusses the methodology of Heuristic search. The algorithm defines three separate modes of operation, namely, NORMAL, OBSTACLE, and OBSTACLE-

EDGE modes, in which different heuristic sets are used to guide the vehicle. Two classes of heuristics exist: the general heuristics which apply under all three modes, and the obstacle clearance heuristics which are operative only when obstacles are encountered during OBSTACLE and OBSTACLE-EDGE modes.

The heuristics serve to prune the otherwise enormous solution space, by selecting only the viable successor sets for further search, thereby, contributing to its speed and versatility. Moreover, since the successor sets all lie within a unit cell of the current vehicle position, the only requirement is for the vehicle sensor to be able detect the obstacles within its close vicinity; thus, complete a priori information on the environment is not required in the case that heuristic search is pursued by a physical agent. Finally, unlike the A* and Best-first search methods, Heuristic search does *not* require an agenda of unexplored paths. This results in efficient computer memory resource usage. The next chapter quantifies its performance relative to the A* and Best-first search strategies.

VI. PATH PLANNING EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter provides a quantitative evaluation of the relative performance of the three path search methods: A*, Best-first, and Heuristic search strategies. In order to highlight the performance of Heuristic search, the paths derived by the three search strategies under the exact same environmental conditions and obstacles are compared.

B. SCENARIOS

Nine different simulation scenarios in the NPS pool environment are defined and used for the study. They are tabulated in Table 6.1 and the detailed definition of each scenario can be found in Appendix A. For each scenario, a different obstacle arrangement or layout is defined in a rectangular boxed region near the center of the pool. In all cases, the Start and Goal positions are located on opposite sides of this obstacle region. The three paths corresponding to the three path-search methods are then derived assuming top-search is preferred, and their results compared.

Scenario 1 evaluates their performance in a clear uncluttered environment. Scenario 2 and 3 tests their ability to find a path around simple obstacles. The remaining scenarios examine their obstacle clearance ability in a randomly cluttered environment. Hence, in the random scenarios (4a through 4f), increasing obstacle densities are defined for the obstacle region. The different densities are simulated by calling a random function software routine with the percentage as an input parameter. This random function then generates obstacles of the specified density in the obstacle region located near the center of the pool. Note that due to the obstacle growing process mentioned in Chapter III, the percentage of virtual obstacles is higher than that specified.

TABLE 6.1 SIMULATION SCENARIOS

SCENARIO	DESCRIPTION
1	No Obstacle
2	Wide wall obstacle
3	High wall obstacle
4a	Region with 5% random obstacles
4b	Region with 10% random obstacles
4c	Region with 15% random obstacles
4d	Region with 20% random obstacles
4e	Region with 25% random obstacles
4f	Region with 30% random obstacles

C. MEASURES OF PERFORMANCE

The search techniques are compared on the basis of the following three quantitative performance measures:

1. Cost of the path
2. Time required to find the path
3. Maximum number of OPEN nodes during the path generation

The cost of a path is the total cost incurred in traversing the path. It is used as a measure of the *optimality* of the path by comparing it with one that has a *minimum* cost. The second and third factors measure the efficiency with which the computer CPU and memory resources are utilized. The OPEN nodes here refer to the leaf nodes of the search tree [Ref. 9]. These two factors are important because with current technology, computing resources are limited.

Note that the algorithms are *not* compared on the basis of the actual value of the quantities, since the latter differs for different implementations as well as in different computers. Rather, it is their relative strengths with respect to each other that are meaningful.

D. RESULTS AND ANALYSIS

1. QUANTITATIVE ANALYSIS

The results of the simulations are summarised and tabulated in Table 6.2 and Table 6.3. Table 6.2 shows the raw data obtained under the defined scenarios, while Table 6.3 summarizes their relative performance with respect to cost.

TABLE 6.2 SIMULATION RESULTS

SCENARIO	FACTORS	HEURISTIC	A*	BESTFIRST
1. No obstacle	Max-open-nodes	1	5432	30
	Time (secs)	0.13	3277	2.1
	Cost	1338	1338	1446
2. Wide wall	Max-open-nodes	1	333	40
	Time (secs)	0.15	9.1	2.5
	Cost	1407	1124	1232
3. High wall	Max-open-nodes	1	3184	23
	Time (secs)	0.15	860	1.28
	Cost	1383	1266	1304
4a. Random 5%	Max-open-nodes	1	2309	22
	Time (secs)	0.12	506	1.14
	Cost	1196	1196	1304
4b. Random 10%	Max-open-nodes	1	724	24
	Time (secs)	0.17	58	1.38
	Cost	1220	1196	1376
4c. Random 15%	Max-open-nodes	1	703	26
	Time (secs)	0.57	53.8	1.60
	Cost	2202	1196	1445
4d. Random 20%	Max-open-nodes	1	309	38
	Time (secs)	0.16	8.95	2.20
	Cost	1407	1148	1256
4e. Random 25%	Max-open-nodes	1	307	38
	Time (secs)	0.16	10.99	2.22
	Cost	1407	1148	1256
4f. Random 30%	Max-open-nodes	1	703	26
	Time (secs)	0.16	14.37	2.20
	Cost	1407	1148	1256

TABLE 6.3 COMPARISONS WITH RESPECT TO PATH COSTS

SCENARIO	A*	HEURISTIC	BESTFIRST	HEURISTIC %DIFF	BESTFIRST %DIFF
1. No obstacles	1338	1338	1446	0.0	8.0
2. Wide wall	1124	1407	1232	25.2	9.6
3. High wall	1266	1383	1304	9.2	3.0
4a. Random 5%	1196	1196	1304	0.0	9.0
4b. Random 10%	1196	1220	1376	2.0	15.0
4c. Random 15%	1196	2202	1445	84.0	20.8
4d. Random 20%	1148	1407	1256	22.5	9.4
4e. Random 25%	1148	1407	1256	22.5	9.4
4f. Random 30%	1148	1407	1256	22.5	9.4

a. Time Required

This quantity measures the raw CPU time required to find a path. Table 6.2 shows that Heuristic search has excellent time performance. In general, Bestfirst search takes about 1 order of magnitude longer, while A* is about 2 orders of magnitude longer than Heuristic search.

b. Maximum Number of Open Nodes

The Heuristic search algorithm has only one OPEN node during the entire search. This is expected since it does not keep an agenda of open nodes to be explored, unlike A* and Best-first search strategies; instead an absolute decision is made at each decision node. This makes Heuristic search highly efficient with reagrd to computer memory usage. A* lies at the other extreme, requiring enormous amount of storage (scenario 1 in Table 6.2) even for such short range scenarios.

c. Cost of Path

The optimality of a path can be measured by the percentage cost difference of its path with respect to the optimal path (i.e. the minimum cost path). In order to show that the A* search algorithm used in this study yields the optimal path, a slight digression is necessary.

A search algorithm is said to be ***admissible*** if, it always terminates in an optimal path from the Start location to the Goal location whenever a path from the Start to the Goal exists. Nilsson [Ref. 9; pp 74 to 79] has shown that in order for the A* search algorithm to be admissible, at any point in the search, the estimated cost of the path from any point in the path to the Goal (as provided by the estimation function), must be less than or equal to the actual cost. This condition is clearly satisfied by the Estimation Function used in this study, since, in fact, the minimum expected cost is used by the function (and there does not exist any path which will give a lower cost).

Thus, the A* search used in this study gives minimum cost paths (that satisfy the constraints of the path specifications), and it can therefore be used as the yardstick for measuring the cost performance of other algorithms.

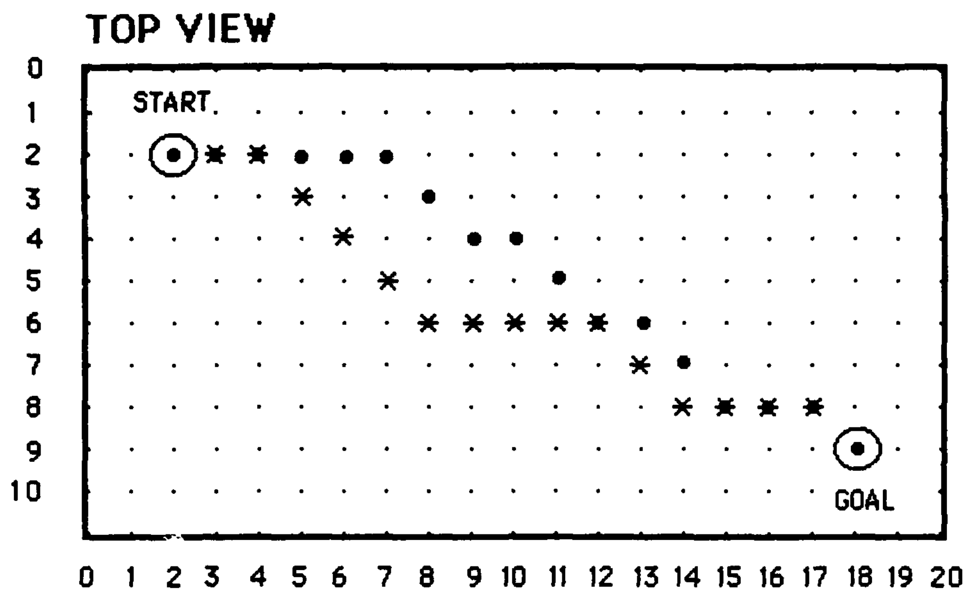
The second last column of Table 6.3 show that, except for scenario 4c, the cost of Heuristic search path is usually within 25% of the optimal path. Its cost performance is optimal or very close to optimal in relatively uncluttered environment (scenarios 1, 4a and 4b). The reason for the high cost in Scenario 4c (differing from optimal by 84%) can be explained by analysing its path - a close analysis of Figure 6.6 reveals that the vehicle path was blocked completely by the wall of the pool and, behaving as a human would, it turned back to find another way through the obstacle. Since autonomous vehicles usually operate in open environments, this situation is an exception rather than the norm. The paths obtained for Best-first search also come within 20% of the optimal solutions as the last column shows.

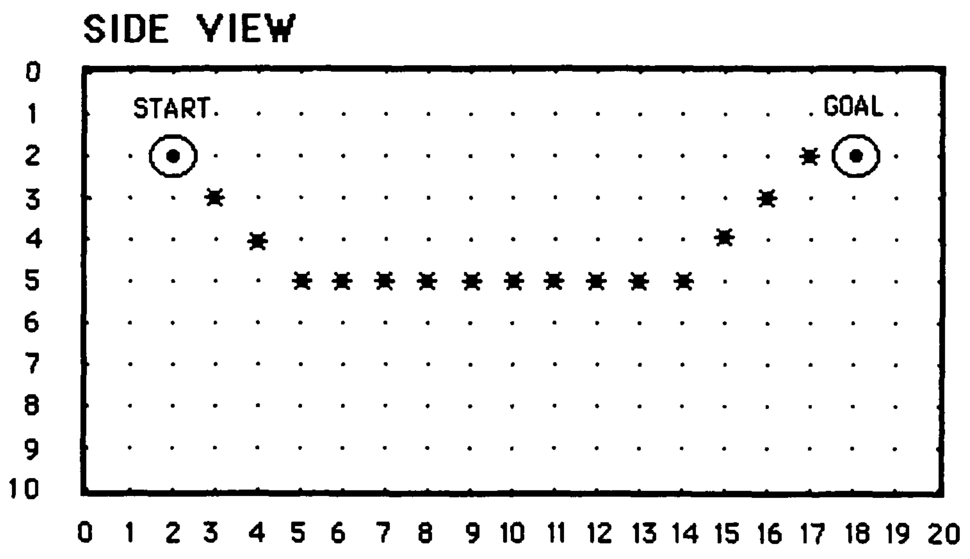
2. QUALITATIVE COMPARISON OF HEURISTIC AND A* PATHS

The paths generated by Heuristic search and A* search for each scenario is shown in Figures 6.1 through 6.9. It can be seen from Figures 6.1 and 6.4 that in clear and uncluttered environments, Heuristic search and A* search yield almost the same paths; in fact, Table 6.2 show that the two paths corresponding to the two methods have the same cost in each case. The slight deviation in the paths, particularly for Scenario 1 (Figure 6.1), is probably due to the fact that the sort routine used in A* search does not preserve the order of the agenda for equal-cost paths. Another contributing factor is that Heuristic search prefers to maintain the vehicle heading as far as possible, and any required changes to its heading are therefore deferred till later in the path.

The paths derived by Heuristic search under Scenarios 2 and 3 (see Figures 6.2 and 6.3), exhibit the characteristic behavior induced by obstacle clearance heuristics, as explained in Sections G and I of Chapter V. In both cases, the Heuristic search path proceeds diagonally along the wall of the obstacle instead of taking the shortest path as established by A* search.

Figures 6.5 through 6.9, reveal that Heuristic search and A* search yield qualitatively very different paths when the environment is increasingly cluttered with obstacles. This is expected since the fundamental strategy of the two methods are different: A* search aims for global optimization while Heuristic search aims for local optimization. It is also noted from Figures 6.7 through 6.9, that as the density of the obstacles increases, the obstacle region becomes effectively a single contiguous block, and the paths derived by either method is independent of the obstacle density.





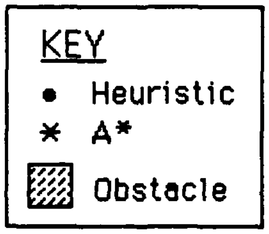
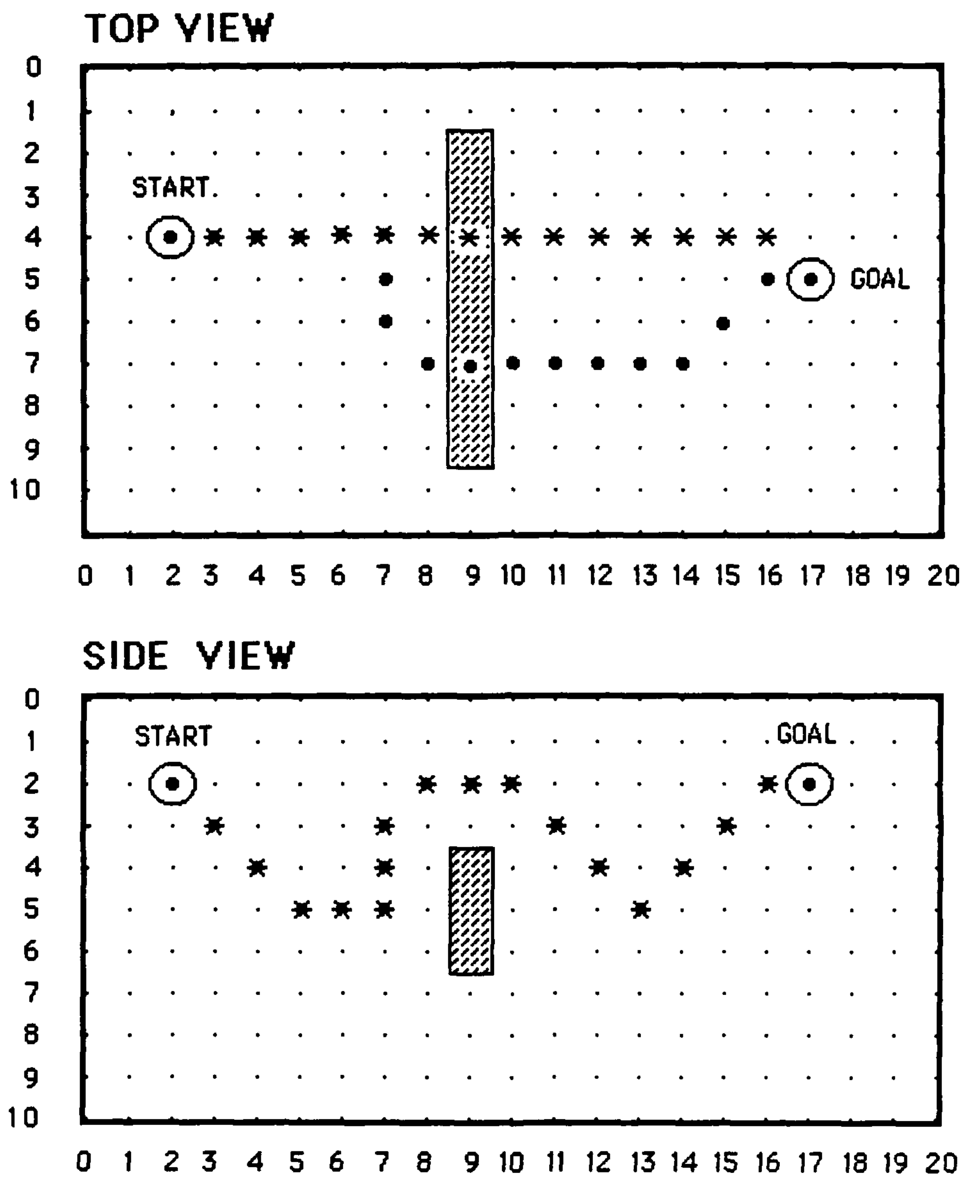


Figure 6.1 Scenario 1: No Obstacle



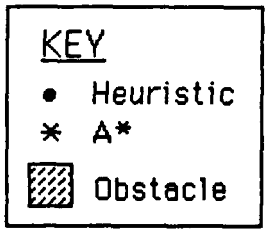
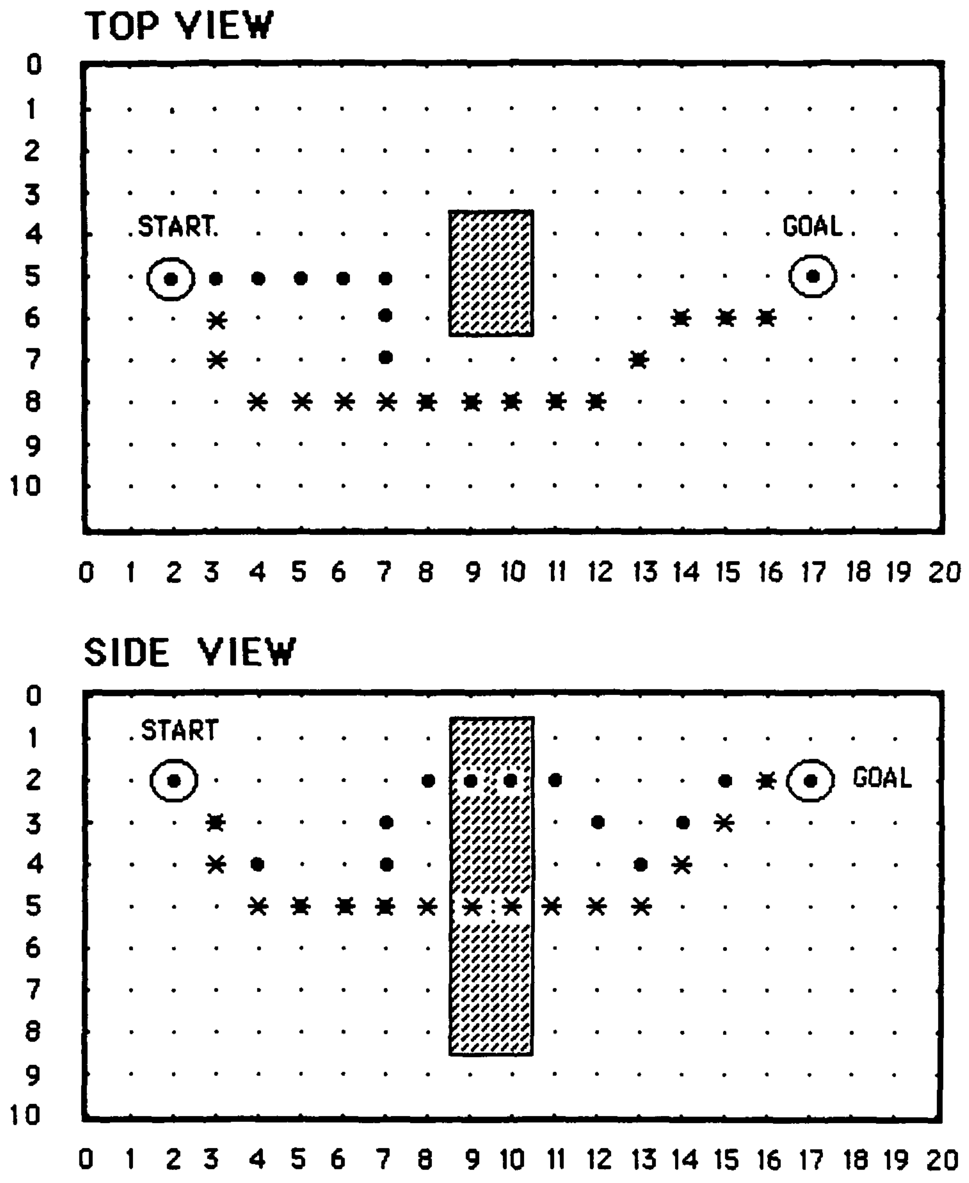


Figure 6.2 Scenario 2: Wide Wall Obstacle



KEY	
●	Heuristic
✱	A*
▨	Obstacle

Figure 6.3 Scenario 3: High Wall Obstacle

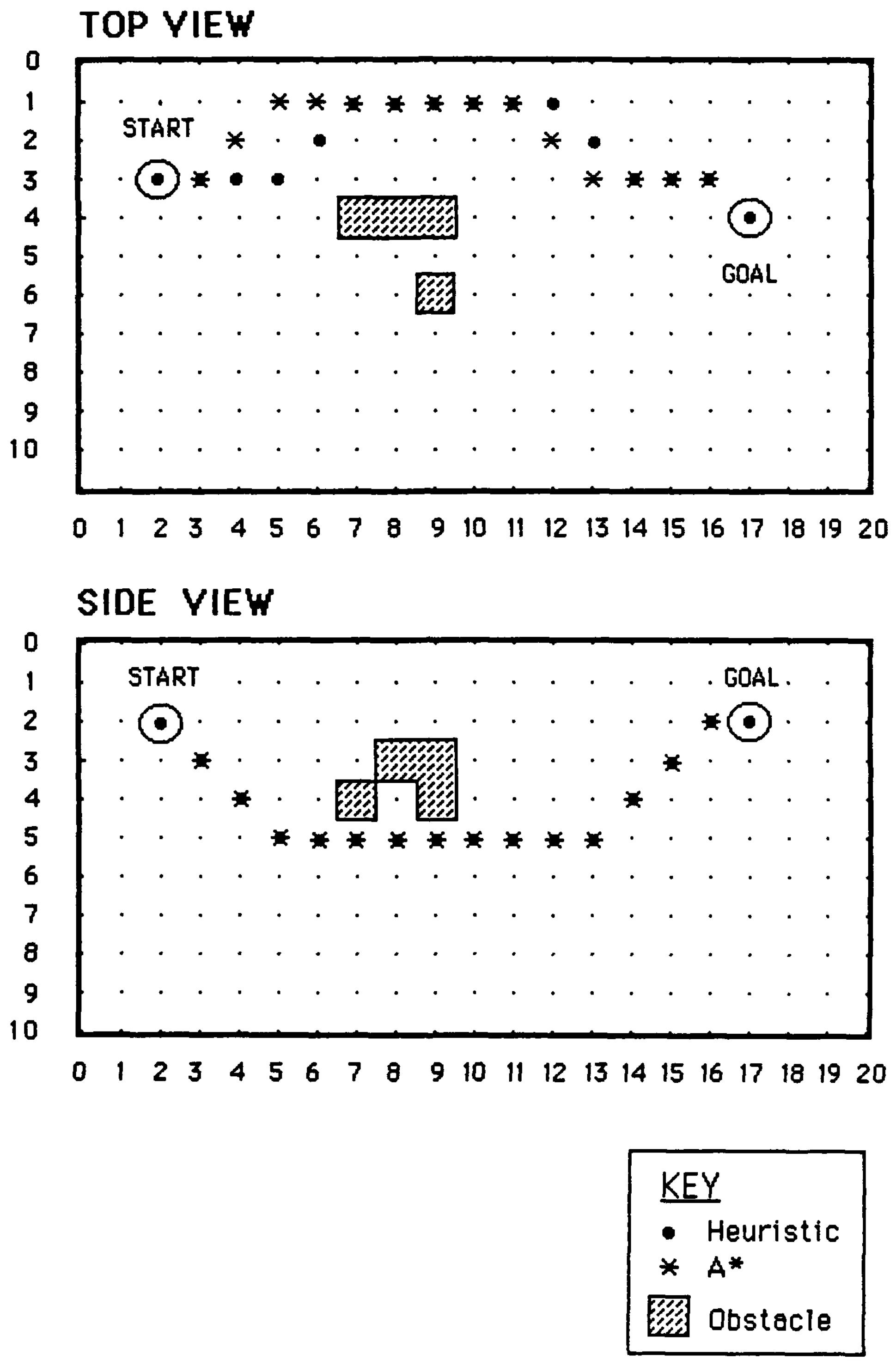


Figure 6.4 Scenario 4a: Random Obstacle 5%

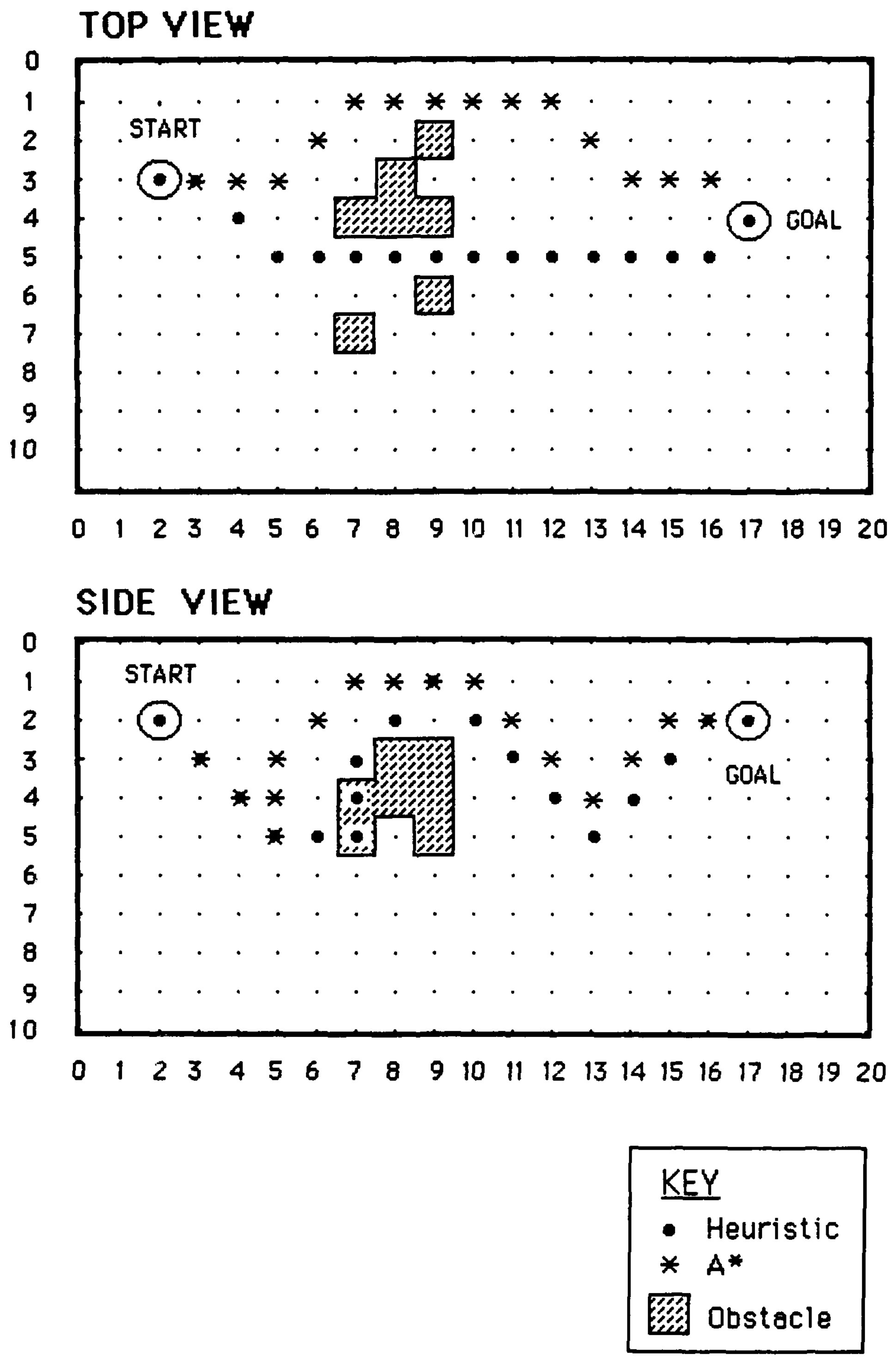


Figure 6.5 Scenario 4b: Random Obstacle 10%

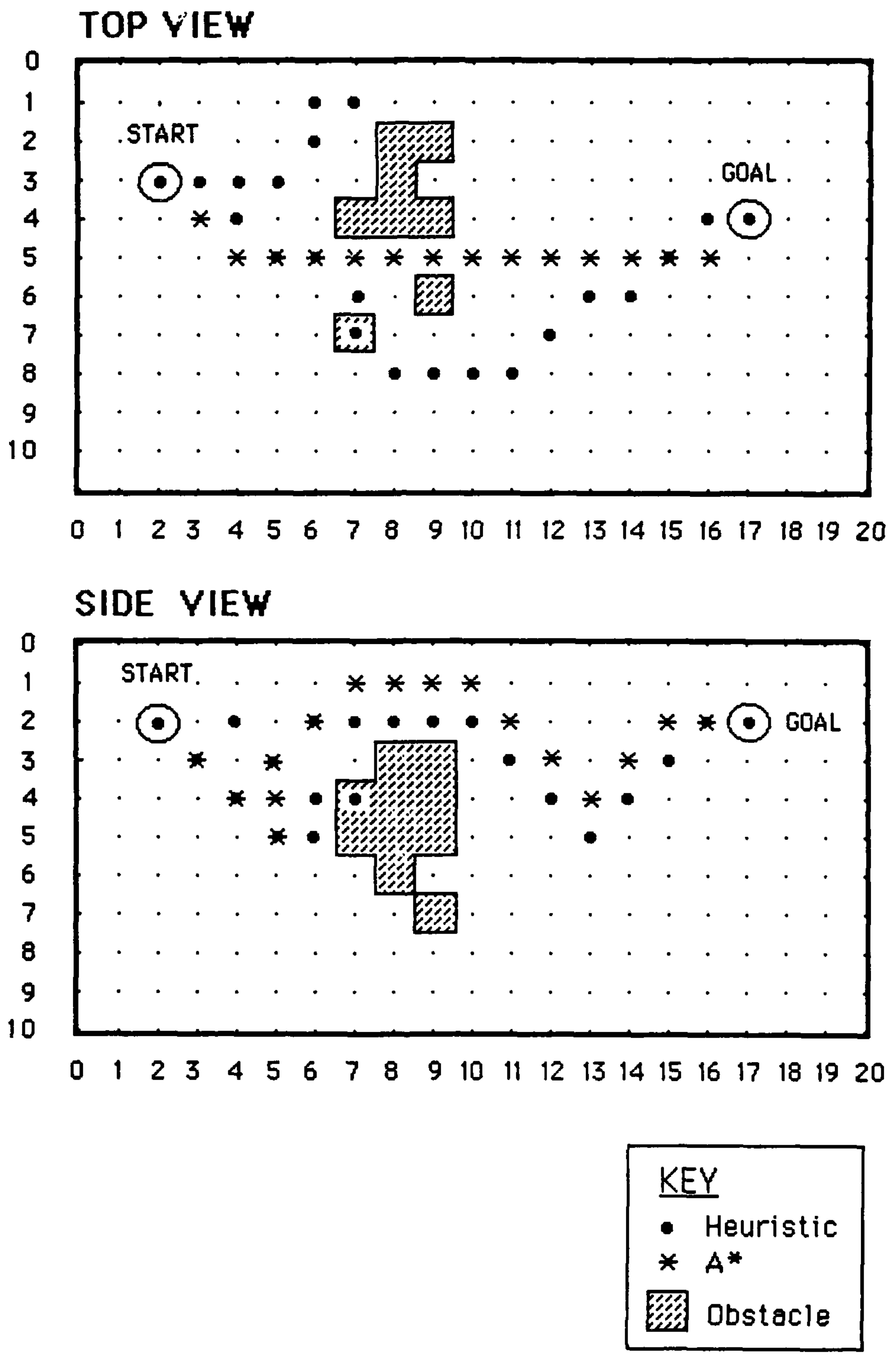
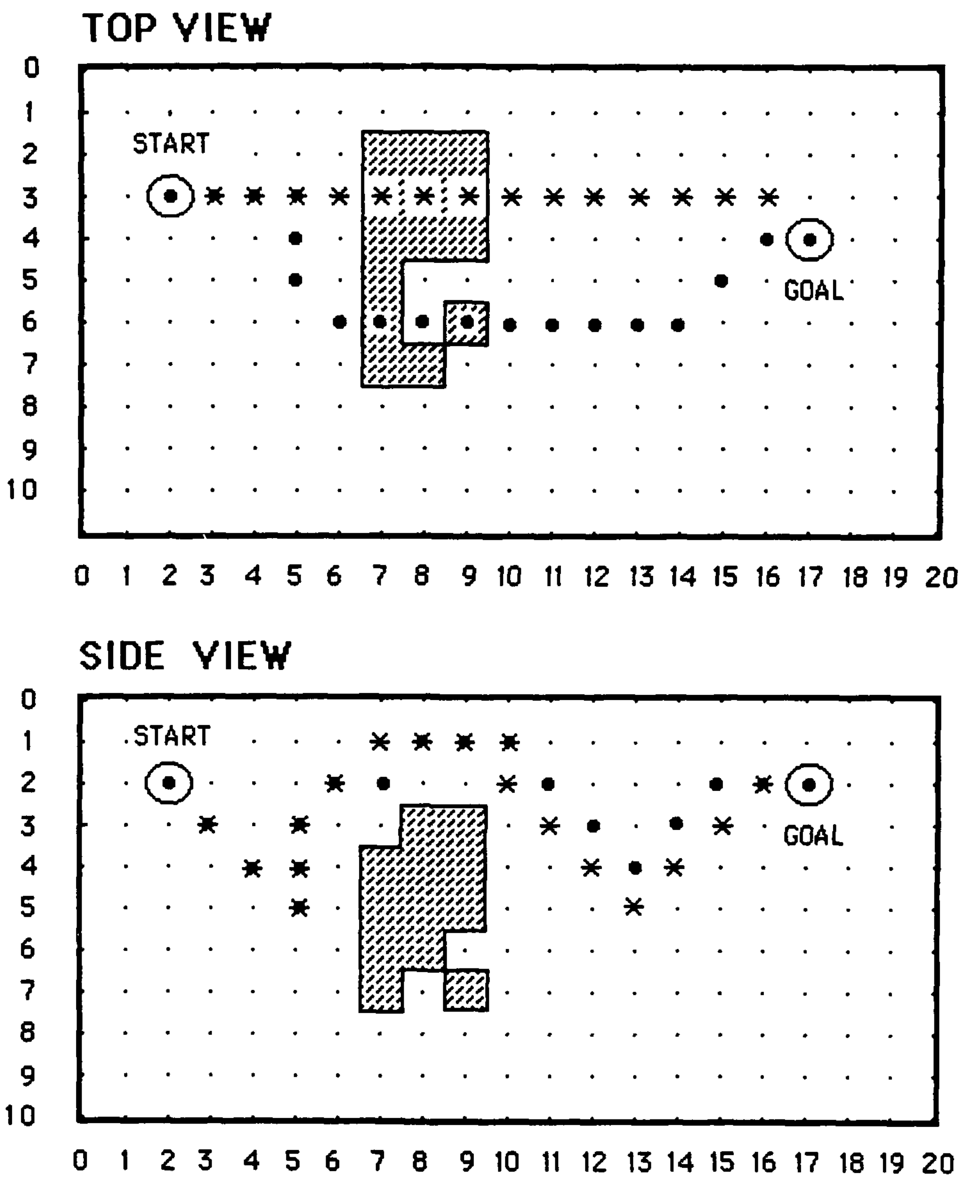


Figure 6.6 Scenario 4c: Random Obstacle 15%



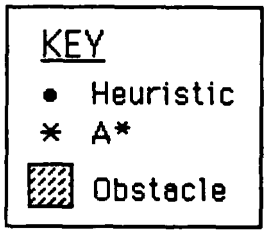
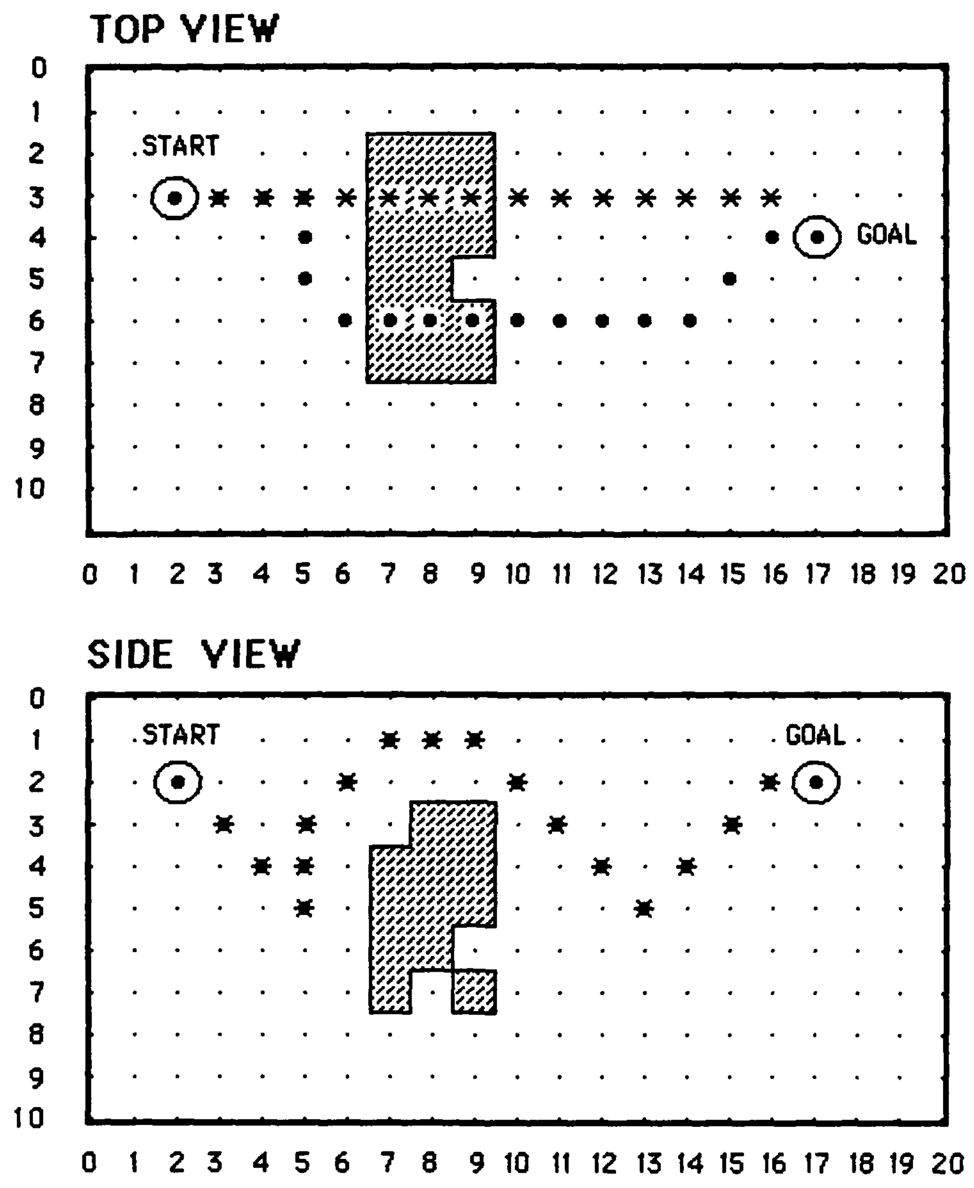


Figure 6.7 Scenario 4d: Random Obstacle 20%



KEY

- ● Heuristic
- ✱ A*
- ▨ Obstacle

Figure 6.8 Scenario 4e: Random Obstacle 25%

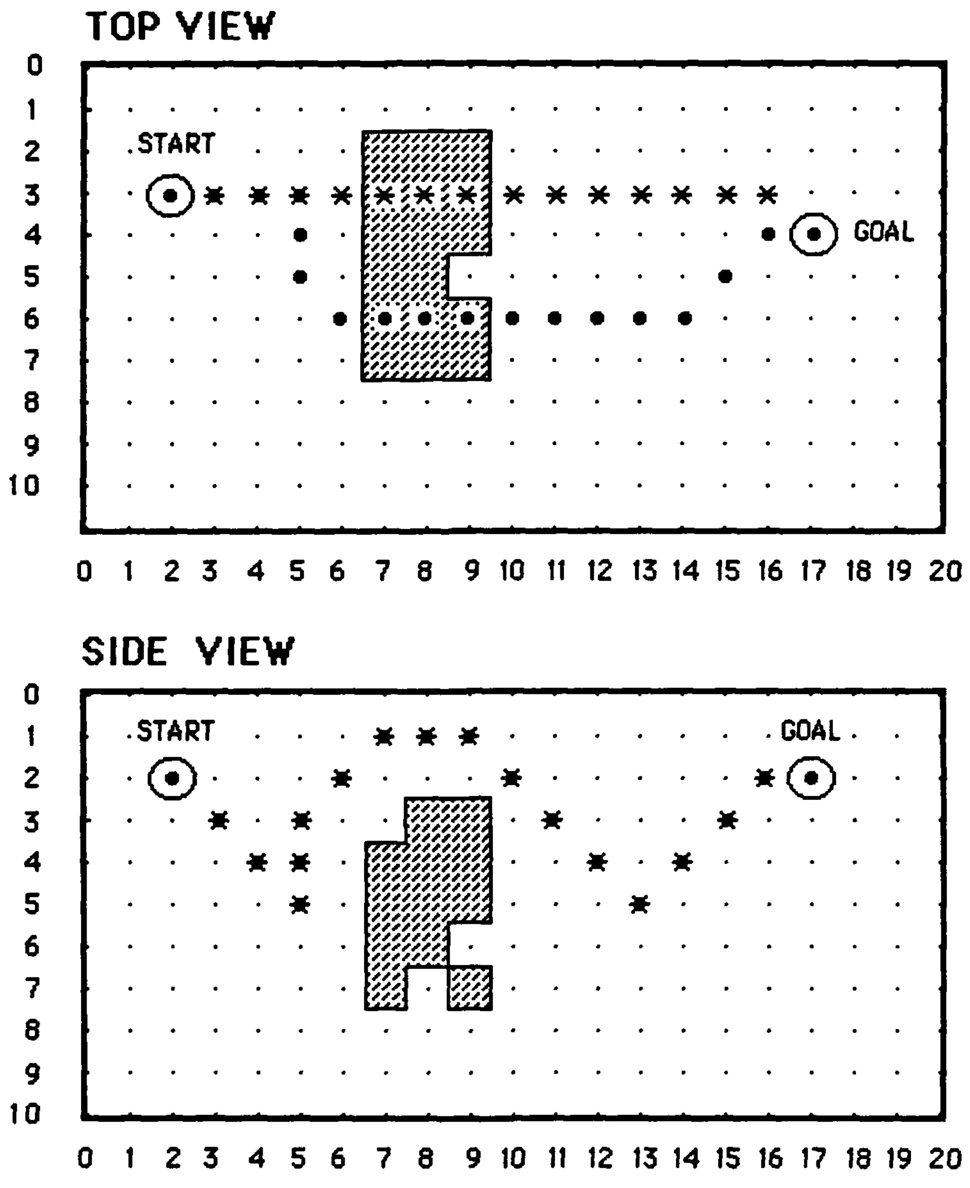


Figure 6.9 Scenario 4f: Random Obstacle 30%

E. SUMMARY

The above results show that Heuristic search is very suitable for autonomous vehicle path-planning where speed of search and space requirements are fundamental considerations, and where the optimality of the path with respect to energy cost is not critical. A* is highly space and time inefficient, but yields an optimal path. The performance of Best-first search lie somewhere between the two. Qualitatively, Heuristic and A* strategies yield almost similar paths for clear and relatively uncluttered regions, and widely different paths for cluttered environments. The most appropriate method to use, therefore, depends on the given mission, and specifically, on the criticality of the time, space and cost constraints.

VII. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

The research efforts under the NPS-AUV program have, thus far, been directed at vehicle design at the Vehicle Control level (Figure 3.2) as well as the creation of the laboratory testbed environment. With the stabilization of the groundwork at this level, current efforts have begun to examine the issues at the higher Mission Planning and Mission Control levels. This thesis represents a step in this direction by addressing the issue of computer support for AUV mission planning. The specific contributions of this thesis are elaborated in the following sections.

1. A PROTOTYPE MISSION PLANNING EXPERT SYSTEM

The MPES serves as an important mission planning aid to human mission planners. The Mission Planning workstation developed provides an informative, easy-to-operate, and a totally interactive control station that allows rapid mission planning and evaluation of plans prior to actual execution. Additionally, with the prototype defined and developed, the MPES can be easily upgraded to handle other more complex missions, as well as providing a basis for experimentation with other rule-bases.

2. SOFTWARE ARCHITECTURE FOR MISSION PLANNING

The software architecture adopted closely models the progressive stages of mission planning. It represents another approach to designing an expert system which automatically transforms the high-level mission specifications to detailed low-level plans. The advantages of this approach are:

1. Simplicity. The complex mission planning task is decomposed into distinct decision-making entities, thereby simplifying its design and development.

2. Flexibility. The incorporation of a centralized control in the design by the Mission Planning Controller provides the flexibility needed to react and adapt to changing situations. This is especially valuable for on-board mission planners and re-planners which have to respond to unexpected events during the execution of the mission.

3. Maintainability and ease of enhancements. Future enhancements to individual entities can be performed with minimum impact to other components. For instance, modifications to the Voters entity either to consider a new constraint or to add a new path-search strategy will only require a new entry to Table 4.1. The new path-search algorithm can also be added to the Mission Constructor with virtually no side-effects on other entities. This attribute is further augmented by exploiting KEE's object oriented and rule-based paradigm - the former facilitates modularity and encapsulation necessary for maintainability, while the latter is suited to the inherently

unstructured nature of the problem.

3. DEVELOPMENT AND ANALYSIS OF ALTERNATIVE THREE-DIMENSIONAL PATH-SEARCH ALGORITHMS

Three path-search strategies, each with differing characteristics were implemented and their performance compared. The results were used as critical inputs to the Mission Planning Expert System.

4. HEURISTIC SEARCH STRATEGY

The Heuristic path-search is developed in this thesis promises to be appropriate for fully autonomous, long-range, high-endurance missions. The speed of this algorithm and its ability to perform without complete a priori environmental information also makes it a practical and viable candidate for real-time on-board path planning functions.

5. GRAPHICAL SIMULATOR UPGRADE

Although it was not intended to be a goal of the research, the code for the IRIS 4D/70GT graphics workstation was upgraded to be on par with the overall status of the NPS-AUV project. This was done in order to provide the framework necessary for more realistic simulations with regard to the test environment and vehicle, prior to the actual in-water tests.

B. RESEARCH EXTENSIONS

There are several broad areas in which future research can be directed. They include the vehicle design, upgrading of the MPES and the testbed simulator, and the on-board Mission Control level functions. The near-term goals, however, should bring about a consolidation of the efforts thus far, and facilitate the construction of a demonstration prototype AUV. The realization of this prototype will serve not only to demonstrate that the myriad ideas and design decisions made are coherent and feasible, but also to uncover early the potential and major design flaws (where they exist). Thus, the immediate research efforts should emphasize the Mission Control level functions and the integration of the hardware and software at the Mission Control and Vehicle Control levels. At the same time, the short and long-term research objectives should set out the blueprint required to realize a fully autonomous AUV, suitable for long-range and high-endurance missions. The following sections describe the possible extensions to four areas: mission planning, mission re-planning, path planning algorithms, and the graphical simulator.

1. MISSION PLANNING

An immediate task is to develop the code necessary for automatic downloading of the planned Mission Details to the on-board Gridcase computer, so as to be ready for the forthcoming in-water demonstration tests. Another near-term goal would be to expand and develop the other mission types in the same fashion as the one developed for the Transit Pool mission. In the process, other new and probably more complex constraints may need to be considered - in particular, constraints such as fuel (energy) requirements and the actual time available for mission execution must be factored in. The off-line database may also require extension to include more realistic open ocean environments although this should be done in conjunction with the upgrades to the graphical simulator to provide the corresponding displays. So far, the missions considered are relatively simple. Thought should also be given to multi-task missions which require optimal task scheduling and more sophisticated route planning capabilities.

2. MISSION RE-PLANNING

A fully autonomous AUV should have the versatility to deviate from original mission plans and to initiate re-planning in response to changed circumstances. For instance, if the mission H-hour has been brought forward, it must be able to re-prioritize and re-plan its tasks in order to achieve the higher mission objectives. Thus, onboard re-planning and re-scheduling capabilities must be incorporated to enable it to respond appropriately. A good starting point would be to modify the off-line mission planning code for the onboard re-planner.

3. PATH PLANNING

Several immediate improvements can be made to the heuristic search algorithm. Firstly, the various cost figures such as rotational cost and translational cost have been estimated in this study; but, with the development of the actual vehicle, more realistic vehicle data should be made available and used.

Secondly, although the heuristic search strategy can theoretically perform without a priori information concerning the environment, it remains to be demonstrated. However, this would not only require changes to the path-planning code (mainly the data structures used for encoding the environment), but it also requires the graphical simulator to be upgraded to simulate processed sonar sensor inputs. The same is true with its capability to deal with dynamically moving obstacles.

The third possible improvement to Heuristic search is the handling of ***concave*** obstacles. This is related to the path-marking feature used to overcome the local minimum problem explained in Section F of Chapter V, since all concave obstacles possess inherent local minimas which may trap the vehicle. Although the path-marking technique can be used, it is inefficient with respect to cost and time in the case where the concave obstacle (or "tunnel") is wide and deep. This is due to the fact that the method has to search almost the entire volume within the concave obstacle before the path marking value becomes sufficiently high to discourage further search within the obstacle. Thus, more elegant and more efficient approaches need to be examined. One promising method that can be explored is the ***obstacle-marking*** technique used in [Ref. 8], for two-dimensional path-planning.

A fourth enhancement to be considered for immediate implementation is the changes required to be performed on the path search algorithms to accomodate the hovering mode of the vehicle. Presently, although the interface for its selection is provided, hovering mode is not considered in the path-search algorithms.

Fifthly, the efficiency of A* search with regard to time and memory resources, can be improved. Specifically, a different sort method can be explored to improve the time required to sort the agenda.

In the longer term, one suggestion for consideration relates to the methodolgy used to realize the heuristics in Heuristic search. Currently, the heuristics are implemented procedurally for execution speed. Another method is to express the heuristics in a higher-level rule form, by using Prolog for instance, although this approach may severely degrade the execution efficiency. Thus, a combination of the rule-based and procedural approaches may be the most effective way to implement the heuristics - a technique which is worth exploring.

Within the Mission Planner, one possible and significant enhancement to the high-level for consideration, is to combine a macro-level route planner with a micro-level path planner. Thus far, the various computer aided prototypes developed in the path planning research community, have dealt solely with one or the other. There is a good possibility, however, for path-planning to be performed in two stages - first, invoking a route planner to derive the major route segments, and then to plan the detailed path (for each path segment) using a path planner.

A final suggestion is to implement and include new path-search strategies to expand the suite of path-planning tools available for the Mission Constructor of the MPES.

4. GRAPHICAL SIMULATOR

The simulator should evolve with the NPS AUV vehicle as well as with the *complexity of the missions* it will undertake. As the physical design of the NPS Model 2 AUV stabilizes and becomes more fully defined, its corresponding hydrodynamic model and maneuvering characteristics should be incorporated on the simulator to validate its performance prior to actual tests. Another important extension, as mentioned above, is the display of open ocean environments to support the testing of more realistic missions. Finally, the simulator can be enhanced to include the *simulation of processed sonar inputs* to be passed to path-planning algorithms. In particular, it can be used to validate the capability of the Heuristic search (and other search strategies) to plan a path without complete a priori environment information.

LIST OF REFERENCES

1. Macpherson, D., *A Computer Simulation Study of Rule-Based Control of an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1988.

2. Nordman, D., *A Computer Simulation Study of Mission Planning and Control for the NPS Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

3. Crowley, J. L., "Navigation for an Intelligent Mobile Robot," *IEEE Journal of Robotics and Automation*, v. RA-1(1), pp. 31-41, 1985.

4. Oommen, B. J. Iyengar, S. S., Rao, S. V. N., and Kashyap, R. L., "Robot Navigation in Unknown Terrains Using Learned Visibility Graphs. Part I: The Disjoint Convex Obstacle Case," *IEEE Journal of Robotics and Automation*, v. RA-3(6), pp. 672-681, 1987.

5. Iyenger, S. S., Jorgensen, C. C., Rao, S. V. N., and Weisbin, C. R., "Learned Navigation Paths for a Robot in Unexplored Terrain," *IEEE Computer Society, The Second Conference on Artificial Intelligence Applications*, pp. 148-155, 1985.

6. Kuan, D. T., Brooks, R. A., Zamiska, J. C., and Das, M., "Automatic Path Planning for a Mobile Robot Using a Mixed Representation of Free Space," *IEEE Computer Society, Conference on Artificial Intelligence Applications*, pp 70-74, 1984.

7. Brooks, R. A., "Solving The Find-Path Problem by Good Rep[illegible]entation of Free Space," *IEEE Transactions on Systems, Man, and Cybernetics*, v. SMC-13, pp 190-197, 1983.

8. Ok, D. K., *A Computer Simulation Study of a Sensor-Based Heuristic Navigation for Three Dimensional Rough Terrain With Obstacles*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

9. Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Co., 1980.

10. Rowe, N. C., *Artificial Intelligence Through Prolog*, Prentice-Hall, Inc., 1988.

11. Elfes, A., "Sonar-Based Real-World Mapping and Navigation", *IEEE Journal of Robotics and Automation*, v. RA-3, No. 3, pp 249-265, 1987.

12. Chappell, S. G., "A Blackboard Based System for Context Sensitive Mission Planning in an Autonomous Vehicle", unpublished technical report, University of New Hampshire, Marine Systems Engineering Laboratory.

13. Blidberg, D. R., and Chappell, S. G., "Guidance and Control Architecture for the EAVE Vehicle," unpublished technical report, Marine Systems Engineering Laboratory, 1986.

14. Russell, G. T., and Lane, D. M., "A Knowledge Based System Framework for Environmental Perception in a Subsea Robotics Context," *IEEE Journal of Oceanic Engineering*, v. OE-11, No. 3, July 1986.

15. Mayer, R., Underbrink, A., Lockledge, J., and Reddy, U., "Situation Based Control Architectures for an AUV," unpublished technical report, Texas A&M University, College Station, Texas.

16. Pugh, G. E., and Krupp J., "The Control of Autonomous Underwater Vehicles through a Hierarchical Structure of Value Priorities", *Proceedings of the Fifth International Symposium on Unmanned, Untethered Submersible Technology, University of New Hampshire*, June 22-24, 1987.

17. Healey, A. J., Papoulias, F. A., Macdonald, G., "Design and Experimental Verification of a Model Based Compensator for Rapid AUV Depth Control," *Proceedings of the 6th Unmanned, Untethered, Submersible Technology Conference*, Washington DC., June 12-14, 1989.

18. Dibble, P., *OS-9 Insights*, Microware Systems Corporation, Des Moines, IA, 1988.

19. Ray, C. R., *A Study of 3-D Visualization and Knowledge-Based Mission Planning and Control for the NPS Model 2 Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

20. Bane, G., and Ferguson, J., "The Evolutionary Development of the Military Autonomous Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Submersible Technology*, v. 5, June 1987.

21. Lozano-Perez, T., and Wesley, M. A., "An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles," *Communications*, v. ACM-22(10), pp.560-570, 1979.

22. Munson, S. A., *Integrated Support for Manipulation and Display of 3D Objects for the Command and Control Workstation of the Future*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

23. *KEE Software Development System User's Manual*, version 3.0, Intellicorp, Mountain View, CA, March 1986.

24. Steele, G. L., *Common LISP*, 2nd Ed.,Digital Press, Bedford, MA, 1990.

25. Keene, S.E., *Object-Oriented Programming in Common Lisp*, Addison-Wesley, Reading, MA, 1989.

26. Richbourg, R. F., *Solving a Class of Spatial Reasoning Problems: Minimal-Cost Path Planning on the Cartesian Plane*, Doctoral Thesis, Naval Postgraduate School, Monterey, CA, June 1987.

27. Brainin, S. M., and McGhee, R. B., "Optimal Biased Proportional Navigation," *IEEE Transactions on Automatic Control*, v. AC-4, No. 4, pp. 440-442, August 1968.

28. Tan, C. H., *A Simulation Study of An Autonomous Steering System for On-Road Operation of Automotive Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1986.

29. Kanayama, Y., Kimura, Y., Noguchi, T., and Miyazaki, F., "A Stable Tracking Control Method for an Autonomous Mobile Robot," *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, May 14-18, 1990.

APPENDIX A

SCENARIO DEFINITIONS

The following Table A.1 provides a detailed definition of each scenario used to analyse the performance of the three search methods, as mentioned in Chapter VI. The parameters that have the same value for all the scenarios are:

Mission Speed = 350 (rpm)

Mission Depth = 50 (inches)

Safety Radius = 350 (inches)

TABLE A.1 DETAILED SCENARIO DEFINITIONS

S/N	Description	Start	Goal	Obstacle Number
1.	No Obstacle	(140 140 20)	(630 1260 20)	0
2.	Wide Wall	(350 140 20)	(350 1190 20)	1
3.	High Wall	(350 140 20)	(350 1190 20)	2
4a.	Random 5%	(210 140 20)	(280 1190 20)	21
4b.	Random 10%	(210 140 20)	(280 1190 20)	22
4c.	Random 15%	(210 140 20)	(280 1190 20)	23
4d.	Random 20%	(210 140 20)	(280 1190 20)	24
4e.	Random 25%	(210 140 20)	(280 1190 20)	25
4f.	Random 30%	(210 140 20)	(280 1190 20)	26

APPENDIX B

AUV TESTBED SIMULATOR USER MANUAL

A. HARDWARE CONFIGURATION

The laboratory AUV testbed simulator is comprised of the following systems:

1. Symbolics 3675 LISP machine
2. Symbolics Color Monitor
3. Silicon Graphics IRIS (SGI) 4D/70GT graphics workstation

The Symbolics LISP machine is directly connected to the Symbolics Color Monitor, and the two together make up the Mission Planning workstation. The former hosts the mission planning software and interfaces with the user via a series of mouse-driven panels, facilitating the interactive input of mission data and the monitoring of the progress of the mission planning cycle; the latter displays the derived path as well as the actual path during the execution phase, in two-dimensional plan and side elevation views of the NPS pool (for the purpose of feedback and evaluation). The SGI graphics workstation is used for 3-D visualization of the vehicle and the environment during a simulated execution of the mission; it comes with a side terminal which is used for starting the program as well as for displaying user prompt messages during the simulation. Communication between the Mission Planning and the SGI graphics workstations is facilitated by an ethernet local area network on which they reside. In this manual, the term *AUV testbed simulator* refers to the complete laboratory testbed configuration, while the *AUV graphics simulator* refers only to the SGI graphics workstation.

B. PRE-REQUISITE FOR USING THIS MANUAL

This manual assumes some basic familiarity with the Symbolics LISP machine and the IRIS graphics workstation. The user is also required to be familiar with the elementary commands in the Unix operating system such as those for login on, traversing the hierarchical directory structure, and simple file manipulations. Finally, some nominal experience with the LISP machine and the KEE expert system shell is required for proper startup and shutdown of the AUV testbed simulator; in-depth knowledge of its operation is not needed.

C. THE SGI GRAPHICS WORKSTATION

The operation of the AUV graphics simulator is described in detail by Ray [Ref. 19]; it is updated and included here both to reflect the changes made and for completeness. The simulation is normally run on the IRIS 4D/70GT, specifically IRIS-5, because of its physical proximity to the LISP workstation, which allows easy viewing of both workstations during the autopilot mode of operation. However, all the IRIS machines are networked in a manner that allows the simulation to be run on either IRIS-1, IRIS-4, or IRIS-5.

1. Start-Up Procedure

To start the simulation, "log on" to the auv account on both the IRIS workstation and the side terminal of the IRIS, and then transfer to the directory */work/auv/ongsm* (where the auv programs reside). Start the simulator program by entering the command *auv* on the side terminal followed by a carriage return. It takes about 10 seconds to initialize and to read in the object data files of the vehicle and the pool, before the graphics is fully displayed on the main IRIS graphics workstation.

2. Display Viewing Controls and Vehicle Controls

When the simulation is started, the right side of the graphical display shows a control panel with a set of *sliders*. This panel provides two types of control: the *display viewing controls* and the *vehicle controls*. The viewing controls are those shown on the top half of the control panel and they are used to alter the viewer's perspective of the display. The vehicle controls, shown on the lower half of the control panel, are used to manually steer the vehicle. All controls shown are activated by using the mouse to manipulate the sliding markers as follows: first, position the cursor at the appropriate slider, then press and hold down the left mouse and drag the marker to the desired new value while still holding down the left mouse. Note that changes to the user's viewpoint using the viewing conrols, should be executed slowly or the user may lose his own perspective in the display.

3. Manual and Autopilot Modes

The simulator can be operated in either the *manual* or the *autopilot* modes. In the manual mode, all vehicle controls shown on the display are active, while in autopilot mode, they are inhibited since the Mission Planning workstation provides the control commands to the vehicle. Note that the display viewing controls are always active. The initial *default mode* is the manual mode; here the simulated vehicle starts on the surface of the pool with a speed of 25 *rpm* on course *east.*

4. Autopilot Mode

The autopilot mode is started by pressing in sequence the ESC-key and the A-key on the main keyboard of the IRIS workstation. Pressing the ESC-key brings the vehicle to the original default starting position, while hitting the A-key puts the system in autopilot mode. After activating the autopilot mode, the side terminal will indicate that the IRIS server is waiting to connect to sym1 (the Symbolics LISP machine) and the following message will prompt the user to start the KEE portion of the simulator to connect the LISP client to the IRIS server:

Ready to commence execution phase
Server waiting to connect to sym1
Server waiting to connect to sym1

The autopilot execution can be interrupted by pressing the Q-key which brings the system back to manual mode; if this is done, the autopilot *cannot* be re-started without exiting the program. Note that the communications sockets must be broken when the program is exited, as explained in the next section.

5. Exiting the Simulation Program

The simulation can be interrupted and the program exited at any point during the autopilot simulation mode by using the pop-up menu. The pop-up menu can be brought up by pressing the right mouse. Selecting the *exit* option on the menu of choices will terminate the display program.

However, the above procedure is still incomplete. One important additional step that must be taken is to break the communications (send and receive) socket connections on both the IRIS workstation and the Symbolics Lisp machine. This must be done on the IRIS server first and then on the Lisp machine. To break the socket connections on the IRIS, go to the side terminal and list the current processes by entering the Unix command *ps*. This brings up a list of active processes together with their corresponding process numbers. Stop any send/receive communication daemons with the *kill <process number>* command. This must then be followed by a corresponding step on the Symbolics Lisp machine, by doing *SELECT-L* to enter the Lisp Listener and issuing the command *(end-con)* to end the "conversation".

Note that this procedure to break communications as described, must be repeated if, on the next activation of the autopilot mode, the system reports that the sockets are already in use. Usually, the procedure is performed not more than twice.

D. THE AUV TESTBED SIMULATOR

1. Initial State of SGI Graphics Workstation and the Symbolics Color Monitor

To start the AUV testbed simulator, the three systems comprising the testbed need to be put in the initial state as follows. First, put the AUV graphics simulator in autopilot mode, by following the steps described in the previous section. This is the initial state for the graphics simulator. Next, ready the Symbolics Color Monitor (of the Mission Planning workstation) by simply depressing the "on" button; if the display is "blurry", press the "degaussing" button and hold it for at least 2 seconds.

2. Initial State of Symbolics Lisp Machine

The next step involves initializing the MPES software on the Symbolics LISP machine. First, "log on" to the machine by first doing a *SELECT-L* to bring up the LISP Listener window (if not already displayed), and then issuing the command *login auv*; this login procedure puts the user in the *auv project account* on the LISP machine.

The next thing to do is to load the *auv-mpes desktop* into the KEE environment; this desktop encapsulates the MPES knowledge-base and reserves the workspace needed by the program. The load procedure is as follows. First, do a *SELECT-K* to get into the KEE environment. Then use the mouse to point the cursor at the desktop icon at the upper left comer of the screen and depress the left mouse button. When a pop-up menu appears, select the *Load Desktop* option. A KEE "typescript" window will appear requesting the name of the desktop to be loaded: enter *sym4:>auv>ongsm>auv-mpes.lisp.* KEE will then respond by first loading the required lisp files and then the MPES knowledge base. (The lisp files contain the lisp functions used by the MPES knowledge base). Loading is completed when a shadow mouse appears on the screen; clicking the left mouse button at this stage will bring up the initial screen of the MPES. The Mission Planning workstation is now ready for operation.

3. Mission Planning and Construction Phases

The procedure for planning a mission during the planning and construction phases, is performed on the Mission Planning workstation as described in Section D of Chapter IV.

4. Mission Execution Phase

The execution phase is started by selecting the *execute* option from the execute-abort panel on the Mission Planning workstation (see Figure 4.6 and Section D of Chapter 4). This selection triggers the establishment of the communications between the IRIS graphics simulator and the Mission Planning workstation. The Mission Planning workstation then initiates a "conversation" with the IRIS, and then sends it the coordinates of the initial position of the vehicle and any obstacles that may be defined for the environment. When the data has been transferred, it issues the following message on the KEE "typescript" window:

Iris5 communication selected.
A conversation with the iris has been initiated.
Connection with iris established.

Initial AUV state sent to iris
Hit a key on Iris5 main terminal to continue

At the same time, the IRIS displays a corresponding message on the side terminal as follows:

Obstacles received from lispmachine
Initial position obtained from lispmachine
Hit any key to receive waypoints from lispmachine

To proceed, hit any key on the main IRIS workstation keyboard. This causes the display program to begin reading in the waypoints sent by the Lisp machine. Note that the transfer of the waypoints from the Lisp machine to the IRIS is not required, and is done only so that the complete path can be displayed on the IRIS for debugging purposes. Upon completion of this transfer, the actual execution then begins.

APPENDIX C
FILE ORGANIZATION

A. IRIS WORKSTATION

Figure C.1 shows the overall hierarchical file organization of the *auv* account on the IRIS1 Workstation. The Iris software used in this thesis are stored in the *share3* and the *ongsm* sub-directories. The following list provides a brief description of the contents of the relevant sub-directories:

1. *share3* -- contains the original IRIS-SYMBOLICS communications software (which is not modified in this thesis).
2. *ongsm* -- contains the current version of the Iris auv software.
3. *symbolics* -- contains a backup of the lisp files for the Symbolics Lisp Machine Workstation.
4. *auvobjs* -- contains the OFF files for the auv graphical objects.
5. *model1objs* -- contains the OFF files for the AUV Model 1 vehicle.
6. *model2objs* -- contains the OFF files for the AUV Model 2 vehicle.

B. SYMBOLICS 3675 LISP MACHINE WORKSTATION

Figure C.2 shows the hierarchical file structure of the auv account on the Symbolics Workstation. The sub-directory *ongsm* contains all the required lisp code, the KEE Knowledge Base, and the KEE Desktop developed for the MPES. Under no circumstances should the contents of this sub-directory be changed or modified.

The *currwork* sub-directory is created as a "working" or "scratch-pad" directory, meant for storing any code that is under development. It is recommended that, prior to any future changes or enhancements that might be made to the auv-mpes lisp code, a copy of the *ongsm* sub-directory contents be made on the *currwork* sub-directory; any modification should then be performed on the *currwork* sub-directory.

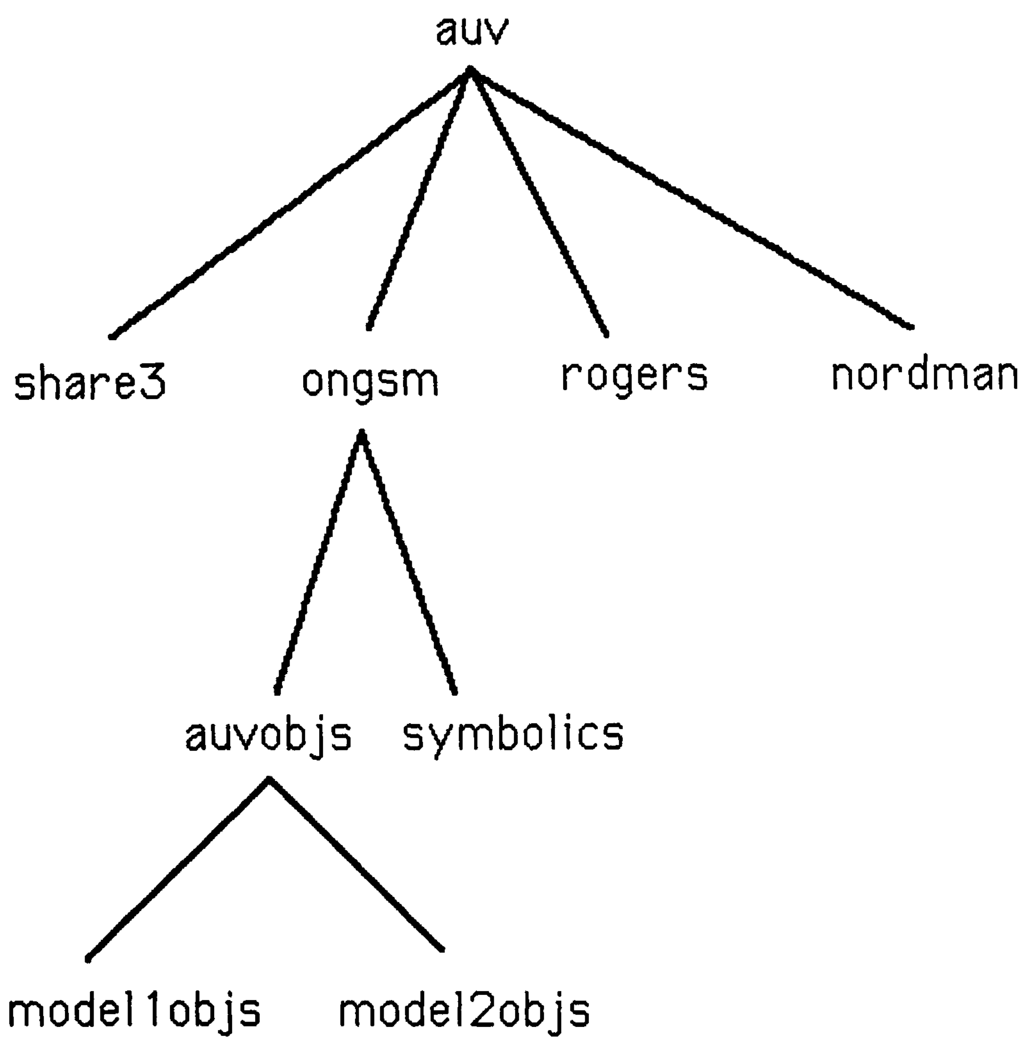


Figure C.1 File Organization on IRIS Workstation

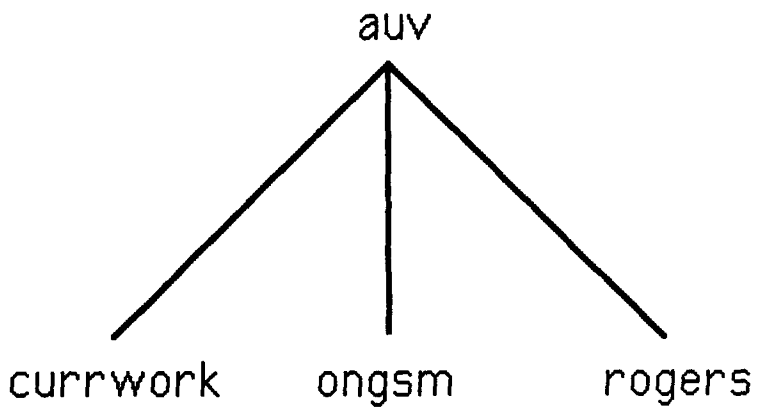


Figure C.2 File Organization on the Symbolics Workstation

APPENDIX D
PROGRAM LIST

This appendix contains the source listings of the lisp code developed for this thesis. They are stored in several files as listed below:

1. array.lisp
2. astar-best.lisp
3. boot.lisp
4. eval.lisp
5. hsearch.lisp
6. monitor.lisp
7. obstacle.lisp
8. posn.lisp
9. succ.lisp
10. sym-iris-comm.lisp
11. missions.lisp
12. mission-agents.lisp
13. umissions.lisp

The last three files, "missions.lisp", "mission-agents.lisp", and "umissions.lisp" contain the methods (lisp functions) referenced directly by the KEE units. In addition to the above list, two other major files exist: *mpexpert.u* and *auv-mpes.lisp*. These two files are automatically generated by KEE when the Knowledge Base and the Desktop (respectively) are created; they are not included here due to their excessive length, and also because it would not benefit the reader of this report.

```
;;; -*- Package: USER; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;************************************************************************************
;
;  Filename.......:   array.lisp
;  Author.........:   Ong Seow Meng
;
;  Date Created...:   18 Aug 1989
;  Description....:   This file contains lisp code that build the evaluation function
;                     map (emap). For convenience, the emap is implemented as a
;                     3-dimensional array corresponding to the pool environment in the
;                     grid-system. Actually a complete 3-dimensional array is not necessary
;                     for keeping track of the obstacle locations (virtual and real) and
;                     for path marking - a two-dimensional map with a pointer to a linked
;                     list of obstacle positions is sufficient (and more storage efficient)
;                     should be considered.
;
;
;  Modifications..:
;
;************************************************************************************


(DEFUN make_emap ()
       (SETF *emap* (MAKE-ARRAY (list (+ *xmapsize* 2)
                                      (+ *ymapsize* 2)
                                      (+ *zmapsize* 2) ) )) )

(DEFUN init_pool_emap ()
       (apply_array_fn #'aelmt_dist_to_goal *emap*
                       1 *xmapsize* 1 *ymapsize* 1 *zmapsize* )
       (mark_emap_boundaries) )

(DEFUN mark_emap_boundaries ()
       ;; x-z plane boundaries
       (apply_array_fn #'aelmt_init_to_infinity
                       *emap* 0 (1+ *xmapsize*) 0 0 0 (1+ *zmapsize*))
       (apply_array_fn #'aelmt_init_to_infinity
                       *emap* 0 (1+ *xmapsize*)
                             (1+ *ymapsize*) (1+ *ymapsize*)
                             0 (1+ *zmapsize*))
       ;; y-z plane boundaries
       (apply_array_fn #'aelmt_init_to_infinity
                       *emap* 0 0 0 (1+ *ymapsize*) 0 (1+ *zmapsize*))
       (apply_array_fn #'aelmt_init_to_infinity
                       *emap* (1+ *xmapsize*) (1+ *xmapsize*)
                             0 (1+ *ymapsize*)
                             0 (1+ *zmapsize*))
       ;; x-y plane boundaries
       (apply_array_fn #'aelmt_init_to_infinity
                       *emap* 0 (1+ *xmapsize*) 0 (1+ *ymapsize*) 0 0)
       (apply_array_fn #'aelmt_init_to_infinity
                       *emap* 0 (1+ *xmapsize*)
                             0 (1+ *ymapsize*)
                             (1+ *zmapsize*) (1+ *zmapsize*)) )


(DEFUN apply_array_fn
       (fname array x-start x-end y-start y-end z-start z-end)
       (DO ((xindex x-start (1+ xindex)))
           ((> xindex x-end))
           (DO ((yindex y-start (1+ yindex)))
               ((> yindex y-end))
               (DO ((zindex z-start (1+ zindex)))
                   ((> zindex z-end))
                   (funcall fname array xindex yindex zindex) ) ) ) )
```

```
(DEFUN aelmt_init_to_zero (array i j k)
       (SETF (AREF array i j k) 0) )

(DEFUN aelmt_init_to_infinity (array i j k)
       (SETF (AREF array i j k) *infinity*) )

(DEFUN aelmt_dist_to_goal (array i j k)
       (SETF (AREF array i j k) (dist_bet_posns (LIST i j k) *goal*)) )
```

```
;;; -*- Mode: LISP; Package: USER; Base: 10; Syntax: Common-lisp -*-

;*****************************************************************************
;
;  Filename.......:  astar-best.lisp
;  Author.........:  Ong Seow Meng
;
;  Date Created...:  Oct 1989
;  Description....:  This file contains the lisp functions for A* and Best-first searches.
;
;  Modifications..:
;
;
;*****************************************************************************

;=============================================================================
;  Functions specific to Best First Search
;=============================================================================

(DEFUN bestfirst.search ()
       (SETF *given-mission-depth* *mission-depth*)
       (SETF *path* (bestfirst_search2 (LIST (LIST *start*)) *goal*)) )


(DEFUN bestfirst_search2 (queue goal-posn)
       (IF (> (LENGTH queue) *max-qlength*) (SETF *max-qlength* (LENGTH queue)))
       (LET* ( (curr-posn (posn (FIRST (FIRST queue))))
               (horiz-dist-to-goal (horiz_coord_dist curr-posn goal-posn)) )
             (IF (<= horiz-dist-to-goal *safety-dist*)
                        (SETF *mission-depth* (z_coord goal-posn))
                        (SETF *mission-depth* *given-mission-depth*) ) )
       (COND ((NULL queue) NIL)
             ((within_vicinityp (posn (CAAR queue)) *goal-vicinity-list*)
                                                      (reverse (FIRST queue)))
             (T (bestfirst_search2 (SORT (APPEND (bestfs_expand_node (FIRST queue)) (REST queue))
                         #'(LAMBDA (path1 path2) (smaller_estimationp path1 path2 goal-posn)) )
                       ;; #'(LAMBDA (path1 path2) (closerp path1 path2 goal-posn)) )
                       goal-posn) ) ) )


(DEFUN bestfs_expand_node (path)
       (eliminate_circular_paths
           (MAPCAR #'(LAMBDA (child) (CONS child path))
                   (remove_obstacle_succs (successorS (FIRST path)) ) ) ) )


(DEFUN smaller_estimationp (path1 path2 goal-posn)
       (LET ( (path1-state (FIRST path1))
              (path2-state (FIRST path2)) )
            (< (bestf_estimation (direction path1-state) (posn path1-state) goal-posn)
               (bestf_estimation (direction path2-state) (posn path2-state) goal-posn) ) ) )


(DEFUN bestf_estimation (succ-dir succ-posn goal-posn)
       (LET ( (coord-dist-to-goal (+ (horiz_coord_dist succ-posn goal-posn)
                                     (abs_vert_coord_dist succ-posn goal-posn)) ) )
            (+ (dist_bet_posns succ-posn goal-posn)
               (COND ((<= coord-dist-to-goal *safety-dist*) 0)
                     (T (rotational_cost succ-dir succ-posn goal-posn)) ) ) ) )
```

```
;================================================================================
;  Functions specific to A-star Search
;
;  The structure of 'path' in Astar Search is:
;      (Mode  eval-fn  cost-so-far  (dir (xn yn zn)) (dir (xn-1 yn-1 zn-1)) .......
;                               ...... (dir (xstart ystart zstart))
;
;================================================================================

(DEFUN astar.search ()
       (SETF *given-mission-depth* *mission-depth*)
       (SETF *path* (astar_search2 (LIST (LIST 'Normal-Mode 0 0 *start*)) *goal*) ) )


(DEFUN astar_search2 (queue goal-posn)
       (IF (> (LENGTH queue) *max-qlength*) (SETF *max-qlength* (LENGTH queue)))
       (LET* ( (curr-posn (posn (FOURTH (FIRST queue))))
               (horiz-dist-to-goal (horiz_coord_dist curr-posn goal-posn)) )
             (IF (<= horiz-dist-to-goal *safety-dist*)
                        (SETF *mission-depth* (z_coord goal-posn))
                        (SETF *mission-depth* *given-mission-depth*) ) )
       (COND ((NULL queue) NIL)
             ((within_vicinityp (posn (FOURTH (FIRST queue))) *goal-vicinity-list*)
                                                  (reverse (CDDDR (FIRST queue))) )
             (T (astar_search2 (SORT (remove_higher_cost_paths
                                         (astar_expand_node (FIRST queue) goal-posn)
                                         (REST queue) )
                                 #'(LAMBDA (path1 path2)
                                         (smaller_evaluationp path1 path2) ) )
                          goal-posn) ) ) )


(DEFUN remove_higher_cost_paths (new-list-of-paths curr-queue)
       ;; removes the higher cost path if two paths lead to the same state.
       (COND ( (NULL new-list-of-paths) curr-queue )
             ( (NULL curr-queue) new-list-of-paths )
             ( T (LET* ( (curr-new-path (FIRST new-list-of-paths))
                         (curr-old-path (FIRST curr-queue))
                         (curr-state-new-path (FOURTH curr-new-path))
                         (curr-state-old-path (FOURTH curr-old-path)) )
                       (IF (EQUAL curr-state-new-path curr-state-old-path)
                           (LET ( (new-path-cost (THIRD curr-new-path))
                                  (old-path-cost (THIRD curr-old-path)) )
                                (COND ( (< new-path-cost old-path-cost)
                                           (CONS curr-new-path
                                              (remove_higher_cost_paths (REST new-list-of-paths)
                                                                         (REST curr-queue) ) ) ;;)
                                      ( T (remove_higher_cost_paths (REST new-list-of-paths)
                                                                      curr-queue )   ) ) )
                           (CONS curr-new-path (remove_higher_cost_paths
                                                 (REST new-list-of-paths) curr-queue )) ) )) ) ))


(DEFUN astar_expand_node (path goal-posn)
       (LET ( (cost-so-far (THIRD  path))
              (curr-state  (FOURTH path)) )
            (SETF *Current-Mode* (FIRST path))
            (LET ( (succ-list (remove_obstacle_succs (successorS curr-state))) )
                 (REMOVE-IF
                    #'(LAMBDA (a_path)
                              (LET ( (path-posn-list (get_path_posn_list (CDDDR a_path))) )
                                   (MEMBER (FIRST path-posn-list) (REST path-posn-list)) ) )

                    (MAPCAR #'(LAMBDA (child)
                                 (APPEND (LIST *Current-Mode*
                                               (astar_evaluation curr-state child cost-so-far)
                          ;;!!                   (astar_evaluation curr-state child goal-posn cost-so-far)
                                                 (+ cost-so-far (astar_delta_cost curr-state
                                                                                  (posn child)) ) )
                                          (CONS child (CDDDR path)) ) )
                         succ-list ) ) ) ) ) )
```

```
(DEFUN astar_evaluation (curr-state succ-state cost-so-far)
       (+ cost-so-far (evaluation curr-state succ-state)) )


(DEFUN astar_delta_cost (curr-state succ-posn)
       (local_cost (direction curr-state) (posn curr-state) succ-posn) )


(DEFUN smaller_evaluationp (path1 path2)
       (< (SECOND path1) (SECOND path2)) )


;=================================================================================
;  Functions shared by both Best-First Search and A-star Search
;=================================================================================

(DEFUN cost_of_path (path-state-list)
       (DO* ( (curr-state-list path-state-list      (REST curr-state-list))
              (curr-state (FIRST  curr-state-list) (FIRST  curr-state-list))
              (next-state (SECOND curr-state-list) (SECOND curr-state-list))
              (curr-dir   (direction curr-state)    (direction curr-state))
              (curr-posn  (posn curr-state)         (posn curr-state))
              (next-posn  (posn next-state)         (posn next-state))
              (total-cost 0 ) )
            ( (NULL (CDDR curr-state-list))
                (+ total-cost (local_cost (direction (FIRST curr-state-list))
                                          (posn (FIRST curr-state-list))
                                          (posn (SECOND curr-state-list)) )) )
            (SETF total-cost (+ total-cost (local_cost curr-dir curr-posn next-posn))) ) )


(DEFUN within_vicinityp (position vicinity-list)
       (COND ( (NULL vicinity-list) NIL)
             ( (EQUAL position (FIRST vicinity-list)) T)
             ( T (within_vicinityp position (REST vicinity-list))) ) )


(DEFUN successorS (curr-state)
       ;; returns the list of successors of the current state according to curr-mode.
       (COND ( (EQUAL *Current-Mode* 'Obstacle-Mode)
                      (IF *DEBUG* (PROGN (PRINC 'Obstacle-Mode) (TERPRI)))
                      (obst_mode_successorS curr-state) )
             ( (EQUAL *Current-Mode* 'Near-Obst-Edge)
                      (IF *DEBUG* (PROGN (PRINC 'Obstacle-Edge) (TERPRI)))
                               (obst_edge_successorS curr-state) )
             ( (EQUAL *Current-Mode* 'Normal-Mode)
                      (IF *DEBUG* (PROGN (PRINC 'Normal-Mode) (TERPRI)))
                               (normal_mode_successorS curr-state) ) ) )


(DEFUN normal_mode_successorS (curr-state)
       (det_search_mode curr-state)
       (LET ( (succ-list (get_succ_list curr-state *search-mode*)) )
            (COND ((all_are_obstaclesp succ-list)
                          (SETF *Current-Mode* 'Obstacle-Mode)
                          (successorS curr-state) )
                  ( T succ-list) ) ) )
```

```
(DEFUN obst_mode_successorS (curr-state)
       ; returns the list of successors of current state and
       ; determiones if ready to proceed to *Near-Obst-Edge*.
       (LET* ( (search-sequence
                      (IF *Bottom-Search-Preferred* *AS-BF-bot-preferred-sequence*
                                                    *AS-BF-top-preferred-sequence*) )
                (first-mode-to-try (FIRST search-sequence)) )

              (LET ( (succ-list
                          (REMOVE-IF  #'(LAMBDA (a-successor) (is_wallp (posn a-successor)))
                                      (get_succ_list curr-state first-mode-to-try) ) ) )
                   (IF (NOT (all_are_obstaclesp succ-list))
                       (PROGN (IF *DEBUG* (PROGN (PRINC 'at-least-one-opening!) (TERPRI)))
                              (IF (OR (no_obstacles_in_succ_listp succ-list)
                                      (depth_threshold_reached (z_coord (posn curr-state))) )
                                      ;; get out of *Obstacle-Mode* if none of the successors
                                      ;; are obstacles.
                                  (SETF *Current-Mode* 'Near-Obst-Edge) )
                              (remove_obstacle_succs succ-list) )
                      ;; else try the rest of sequence but remain in Obstacle-Mode.
                      (DO* ( (curr-seq-ls (REST search-sequence) (REST curr-seq-ls))
                              (curr-smode  (FIRST curr-seq-ls)  (FIRST curr-seq-ls))
                              (succ-list (get_succ_list curr-state curr-smode)
                                         (get_succ_list curr-state curr-smode) ) )
                           ( (NOT (all_are_obstaclesp succ-list))
                                     (PROGN (IF *DEBUG* (PROGN (PRINC 'returned_succ_list_ )
                                                               (PRINC succ-list) (TERPRI) ))
                                            (remove_obstacle_succs succ-list) ) ) ) ) ) ) )


(DEFUN obst_edge_successorS (curr-state)
       (LET* ( (fwd-level-succ-list (fwd_level_succ_list curr-state))
                (toward-mission-depth-succ-list (IF (< (depth curr-state) *mission-depth*)
                                                     (fwd_dive_succ_list  curr-state)
                                                     (fwd_rise_succ_list  curr-state) ))
                (total-succ-list
                        (REMOVE-IF  #'(LAMBDA (a-successor) (is_wallp (posn a-successor)))
                          (APPEND fwd-level-succ-list toward-mission-depth-succ-list)) ) )
              (COND ( (no_obstacles_in_succ_listp total-succ-list)
                                        (SETF *Current-Mode* 'Normal-Mode)
                                        (successorS curr-state) )
                    ( (all_are_obstaclesp total-succ-list)  (SETF *Current-Mode* 'Obstacle-Mode)
                                                            (successorS curr-state) )
                    ( T (IF (NOT (all_are_obstaclesp fwd-level-succ-list))
                            fwd-level-succ-list
                            toward-mission-depth-succ-list )) ) ) )


(DEFUN all_are_obstaclesp (succ-list)
       (COND ( (NULL succ-list) T)
             ( (NOT (is_obstaclep (posn (FIRST succ-list)))) NIL)
             ( T (all_are_obstaclesp (REST succ-list))) ) )

(DEFUN get_path_posn_list (path-state-list)
       (COND ( (NULL path-state-list) NIL )
             ( T (CONS (posn (FIRST path-state-list))
                        (get_path_posn_list (REST path-state-list)) )) ) )

(DEFUN remove_obstacle_succs (succ-list)
       (REMOVE-IF #'(LAMBDA (cand-succ-state)
                            (is_obstaclep (posn cand-succ-state)) )
                  succ-list ) )

(DEFUN eliminate_circular_paths (list-of-paths)
       (REMOVE-IF
           #'(LAMBDA (a_path)
                     (LET ( (path-posn-list (get_path_posn_list a_path)) )
                          (MEMBER (FIRST path-posn-list) (REST path-posn-list)) ) )
           list-of-paths) )
```

```
;;; -*- Package: USER; Mode: LISP; Base: 10; Syntax: Common-Lisp -*-

;*************************************************************************************
;
;  Filename.......:  boot.lisp
;  Author.........:  Ong Seow Meng
;
;  Date Created...:  27 Dec 1989
;  Description....:  This file contains the global variable and global constant
;                    definitions as well as the parameter-initialization routines.
;
;  Modifications..:
;
;*************************************************************************************

;------------------------------------------------------------------------------------
;  GLOBAL VARIABLES
;------------------------------------------------------------------------------------

(DEFVAR *DEBUG* NIL)

(DEFVAR *goal*)
(DEFVAR *start*)
(DEFVAR *mission-depth*)
(DEFVAR *given-target-depth*)
(DEFVAR *safety-dist*)
(DEFVAR *path*)
(DEFVAR *real-path*)
(DEFVAR *return-path*)
(DEFVAR *goal-vicinity-list*)
(DEFVAR *curr-speed*)
(DEFVAR *vert-mvt-speed*)
(DEFVAR *vert-turning-speed*)
(DEFVAR *iris-sym-comms-established*)     ;; set by send_parameters_to_IRIS
                                          ;; reset by abort_mission method of EXECUTOR

(DEFVAR *emap*)
(DEFVAR *xmapsize*)
(DEFVAR *ymapsize*)
(DEFVAR *zmapsize*)

(DEFVAR *ObstacleLs*)
(DEFVAR *NumObstacles*)

(DEFVAR *risk-factor*)
(DEFVAR *real-horiz-dist-pu-coord* 70.0)
(DEFVAR *real-vert-dist-pu-coord*  10.0)
(DEFVAR *up-costpu-dist*)
(DEFVAR *down-costpu-dist*)
 DEFVAR *approx-half-real-horiz-dist-pu-coord*)

(DEFVAR *Bottom-Search-Preferred*)
(DEFVAR *top-preferred-sequence*)
(DEFVAR *bottom-preferred-sequence*)

(DEFVAR *search-mode*)
(DEFVAR *Obstacle-Mode*)
(DEFVAR *Near-Obst-Edge*)
```

```
;-----------------------------------------------------------------------------------
;  GLOBAL CONSTANTS
;-----------------------------------------------------------------------------------

;; Constants used in path search programs
(DEFCONSTANT *PI* 3.142)
(DEFCONSTANT *half-PI* (/ *PI* 2))
(DEFCONSTANT *one-eight-PI* (/ *PI* 8))
(DEFCONSTANT *infinity* 100000)
(DEFCONSTANT *deg-to-rad-factor* (/ *PI* 180.0))
(DEFCONSTANT *rad-to-deg-factor* (/ 180.0 *PI*))



;-----------------------------------------------------------------------------------
;  LOADFILES loads all the files containing the required lisp functions
;-----------------------------------------------------------------------------------

(DEFUN loadfiles ()
       (load "sym4:>auv>currwork>array")
       (load "sym4:>auv>currwork>obstacle")
       (load "sym4:>auv>currwork>posn")
       (load "sym4:>auv>currwork>succ")
       (load "sym4:>auv>currwork>eval")
       (load "sym4:>auv>currwork>hsearch")
       (load "sym4:>auv>currwork>astar-best")
       (load "sym4:>auv>currwork>monitor")
       (load "sym4:>auv>currwork>sym-iris-comm")
       (load "sym4:>auv>currwork>umissions")
       (load "sym4:>auv>currwork>missions")
       (load "sym4:>auv>currwork>mission-agents") )


;-----------------------------------------------------------------------------------
; INIT_SEARCH_PARAMETERS function initializes the system parameters used for path search.
;-----------------------------------------------------------------------------------

  (DEFUN init_search_parameters ()
         (IF *DEBUG* (FORMAT T "~%   Entered function 'init_system_parameters'."))
      ;; (SETF *risk-factor* 90)
         (SETF *risk-factor* 0)
         (SETF *vert-mvt-speed* 200.0)
         (SETF *vert-turning-speed* 200.0)
         (SETF *real-horiz-dist-pu-coord* 70.0)
         (SETF *real-vert-dist-pu-coord*  10.0)
         (SETF *approx-half-real-horiz-dist-pu-coord*
               (- (/ *real-horiz-dist-pu-coord* 2) 0.1))
         (SETF *up-costpu-dist* 1.2)
         (SETF *down-costpu-dist* 1.2)
         (SETF *top-preferred-sequence* '(fwd-rise fwd-level top-rl fwd-top top
                                          fwd-dive bot-rl fwd-bot right-left back-up) )
         (SETF *bottom-preferred-sequence* '(fwd-dive fwd-level bot-rl fwd-bot bottom
                                             fwd-rise top-rl fwd-top right-left back-up) )
         (SETF *AS-BF-top-preferred-sequence* '(fwd-rise-and-level  top-all-and-rl
                                                fwd-dive  bot-all  back-up) )
         (SETF *AS-BF-bot-preferred-sequence* '(fwd-dive-and-level  bot-all-and-rl
                                                fwd-rise  top-all  back-up) ) )


(init_search_parameters)
(loadfiles)
```

```
;;; -*- Package: USER; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;*****************************************************************************************
;
;  Filename.......:   eval.lisp
;  Author.........:   Ong Seow Meng
;
;  Date Created...:   18 Aug 1989
;  Description....:   This file contains the lisp code for computing the evaluation
;                     function and it component costs, used in Heuristic search.
;
;  Modifications..:
;
;
;*****************************************************************************************

(DEFUN evaluation (curr-state succ-state)
     ;; returns the evaluation function (F) in MAIN mode.
     ;;        F = C + E
     ;;            where C = local_cost function
     ;;                  E = estimation function
     (+ (local_cost (direction curr-state) (posn curr-state)
                    (posn succ-state) )
        (estimation (direction succ-state) (posn succ-state) ) ) )

(DEFUN estimation (succ-dir succ-posn)
     ;; returns the estimation function
     ;;  E = T(CPn+1,Pgoal) + PM(CPn+1) + R(CPn+1,Pgoal)
     ;;      where (T + PM) is stored in estimation map, emap
     ;; + translation_cost  succ-posn *goal*
     (+ (AREF *emap* (x_coord succ-posn)
                     (y_coord succ-posn)
                     (z_coord succ-posn) )
        (rotational_cost succ-dir succ-posn *goal*) ) )

(DEFUN local_cost (curr-dir curr-posn tgt-posn)
     ;; returns the local cost C = T + R
     ;;    where  T = translational cost
     ;;           R = rotational cost
     (+ (translation_cost curr-posn tgt-posn)
        (rotational_cost  curr-dir curr-posn tgt-posn)
        (risk_cost tgt-posn) ) )

(DEFUN risk_cost (candidate-posn)
      (* *risk-factor* (depth_to_go candidate-posn)) )
     ;; (LET ( (dist-from-start (horizontal_dist candidate-posn (posn *start*)))
     ;;        (dist-from-goal  (horizontal_dist candidate-posn *goal*)) )
     ;;     (COND ((<= dist-from-goal *safety-dist*)
     ;;                 (SETF *mission-depth* (z_coord *goal*)) ) )


(DEFUN depth_to_go (posn)
       ;; (LET ((curr-depth (- *zmapsize* (z_coord posn))))
       (ABS (- *mission-depth* (z_coord posn))) )

(DEFUN translation_cost (curr-posn tgt-posn)
     ;; returns the translational cost (T) from curr-posn to tgt-posn.
     (+ (depth_change_cost curr-posn tgt-posn)
        (horizontal_dist   curr-posn tgt-posn) ) )

(DEFUN depth_change_cost (posn1 posn2)
     ;; returns the cost of changing depth in moving
     ;; from posn1 to posn2.
     (LET ((vert-dist (vertical_dist posn1 posn2)))
          (COND ((= vert-dist 0) 0)
                ((> vert-dist 0) (* *up-costpu-dist* (ABS vert-dist))) ;; moving up towards surface
                ((< vert-dist 0) (* *down-costpu-dist* (ABS vert-dist))) ) ) )
```

```
(DEFUN rotational_cost (curr-dir curr-posn tgt-posn)
     ;; returns the turning cost (RC) in moving from curr-posn to tgt-posn
     (*  *real-horiz-dist-pu-coord*
        (COND ((= (horiz_coord_dist curr-posn tgt-posn) 0) 0)
              (t (LET* ((new-dir   (azimuth curr-posn tgt-posn))
                        (abs-delta (ABS (turn_angle curr-dir new-dir)))
                        (turn-quantum (/ abs-delta 0.3926991)) )
                        ; turn-quantum is in units of 22.5deg.
                      ;; (PRINC 'turn-quantum_= ) (PRINC turn-quantum) (TERPRI)
                       (COND ((rangep turn-quantum  0  1)  0)
                             ((rangep turn-quantum  1  3)  0.1)
                             ((rangep turn-quantum  3  5)  0.5)
                             ((rangep turn-quantum  5  7)  1.0)
                             ((rangep turn-quantum  7  9)  2.0)
                             ((rangep turn-quantum  9 11)  1.0)
                             ((rangep turn-quantum 11 13)  0.5)
                             ((rangep turn-quantum 13 15)  0.1)
                             ((>      turn-quantum 15   )  0) ) )) ) ) )


(DEFUN rangep (var lower upper)
       (IF (AND (>= var lower) (< var upper))  t  NIL) )

(DEFUN azimuth (from-posn to-posn)
       ;; azimuth is the angle in x-y plane with zero along the y-axis
       (ATAN (xcoord_diff from-posn to-posn)
             (ycoord_diff from-posn to-posn) ) )

(DEFUN turn_angle (azimuth1 azimuth2)
       (- azimuth2 azimuth1) )

(DEFUN dir_quantum (azimuth)
       (/ azimuth *one-eight-PI*) )

(DEFUN dist_bet_posns (posn1 posn2)
       (SQRT (+ (sqr (horizontal_dist posn1 posn2))
                (sqr (vertical_dist posn1 posn2)) )) )

(DEFUN horiz_coord_dist (posn1 posn2)
       (SQRT (+ (sqr (abs_xcoord_diff posn1 posn2))
                (sqr (abs_ycoord_diff posn1 posn2))) ) )

(DEFUN abs_vert_coord_dist (posn1 posn2)
       (abs_zcoord_diff posn1 posn2) )

(DEFUN horizontal_dist (posn1 posn2)
       (* *real-horiz-dist-pu-coord* (horiz_coord_dist posn1 posn2)) )

(DEFUN vertical_dist (posn1 posn2)
       (* *real-vert-dist-pu-coord* (zcoord_diff posn1 posn2) ) )

(DEFUN sqr (n)
       (* n n) )

(DEFUN abs_xcoord_diff (posn1 posn2)
       (ABS   (- (x_coord posn2) (x_coord posn1))) )

(DEFUN abs_ycoord_diff (posn1 posn2)
       (ABS   (- (y_coord posn2) (y_coord posn1))) )

(DEFUN abs_zcoord_diff (posn1 posn2)
       (ABS   (- (z_coord posn2) (z_coord posn1))) )
```

```
(DEFUN xcoord_diff (posn1 posn2)
       (- (x_coord posn2) (x_coord posn1)) )

(DEFUN ycoord_diff (posn1 posn2)
       (- (y_coord posn2) (y_coord posn1)) )

(DEFUN zcoord_diff (posn1 posn2)
       (- (z_coord posn2) (z_coord posn1)) )

(DEFUN x_coord (posn)
       (CAR posn) )

(DEFUN y_coord (posn)
       (CADR posn) )

(DEFUN z_coord (posn)
       (CADDR posn) )
```

```
;;; -*- Package: USER; Syntax: Common-Lisp; Base: 10; Mode: LISP -*-

;*******************************************************************************
;
;  Filename.......:  hsearch.lisp
;  Author.........:  Ong Seow Meng
;
;  Date created...:  20 Aug 1989
;  Description....:  This file contains the lisp code for heuristic search.
;
;  Notes..........:
;                    A State is defined by the list-form
;                         (course (x y z))
;                     where (x y z) is a path planning coord.
;                     and course is north in the positive Y-axis.
;
;
;  Modifications..:
;
;*******************************************************************************


(DEFUN heuristic.search ()
       (SETF *path* (heuristic_search2 *start*)) )


(DEFUN heuristic_search2 (curr-state)
       (LET* ( (curr-posn (posn curr-state))
               (horiz-dist-to-goal (horiz_coord_dist curr-posn *goal*)) )
             (IF (<= horiz-dist-to-goal *safety-dist*)
                  (SETF *mission-depth* (z_coord *goal*)) )
             (COND ( (within_vicinityp curr-posn *goal-vicinity-list*) (LIST curr-state) )
                   ( T (LET ((succ-state (successor curr-state)))
                            (path_mark curr-state succ-state)
                            (APPEND (LIST curr-state) (heuristic_search2 succ-state)) )) ) ) )


(DEFUN successor (curr-state)
       ;; returns the best successor of the current state.
       (COND ( *Obstacle-Mode*  (IF *DEBUG* (PROGN (PRINC 'Obstacle_Mode) (TERPRI)))
                                (get_obst_mode_successor curr-state) )
             ( *Near-Obst-Edge* (IF *DEBUG* (PROGN (PRINC 'Obstacle_Edge) (TERPRI)))
                                (get_obst_edge_successor curr-state) )
             ( T (det_search_mode curr-state)
                 (get_normal_mode_successor curr-state
                      (get_succ_list curr-state *search-mode*)) ) ) )


(DEFUN get_obst_edge_successor (curr-state)
       (LET* ( (fwd-level-succ-list (fwd_level_succ_list curr-state))
               (toward-mission-depth-succ-list (IF (< (depth curr-state) *mission-depth*)
                                                   (fwd_dive_succ_list  curr-state)
                                                   (fwd_rise_succ_list  curr-state) ))
               (total-succ-list
                    (REMOVE-IF #'(LAMBDA (a-successor) (is_wallp (posn a-successor)))
                      (APPEND fwd-level-succ-list toward-mission-depth-succ-list)) ) )
             (COND ( (no_obstacles_in_succ_listp total-succ-list)
                                   (SETF *Near-Obst-Edge* NIL)
                                   (successor curr-state) )
                   ( (all_are_obstaclesp total-succ-list) (SETF *Near-Obst-Edge* NIL)
                                                          (SETF *Obstacle-Mode* T)
                                                          (successor curr-state) )
                   ( T (LET ( (best-succ (get_best_succ curr-state fwd-level-succ-list)) )
                            (IF (not_obstaclep (posn best-succ))
                                best-succ
                                (get_best_succ curr-state toward-mission-depth-succ-list) ))) ) ) )
```

```
(DEFUN is_wallp (position)
       (COND ( (OR (= (x_coord position) *xmapsize*) (= (x_coord position) 0)) T )
             ( (OR (= (y_coord position) *ymapsize*) (= (y_coord position) 0)) T )
             ( (OR (= (z_coord position) *zmapsize*) (= (z_coord position) 0)) T )
             ( T NIL) ) )


(DEFUN get_obst_mode_successor (curr-state)
       (LET* ( (search-sequence
                      (IF *Bottom-Search-Preferred* *bottom-preferred-sequence*
                                                    *top-preferred-sequence*) )
               (first-mode-to-try (FIRST search-sequence)) )

             (IF (is_wallp (posn (FIRST (get_succ_list curr-state first-mode-to-try))))
                 (SETF first-mode-to-try (SECOND search-sequence)) )

             (LET* ( (succ-list (get_succ_list curr-state first-mode-to-try))
                     (best-succ (get_best_succ curr-state succ-list)) )

                  (IF (NOT (is_obstaclep (posn best-succ)))
                      (PROGN (IF *DEBUG* (PROGN (PRINC 'found-a-way!) (TERPRI)))
                             (IF *DEBUG* (PROGN (PRINC 'first_try_best-succ_is_ )
                                          (PRINC best-succ) (TERPRI) ) )
                             (IF (OR (NOT (at_least_one_obstaclep succ-list))
                                     (depth_threshold_reached (z_coord (posn best-succ))) )
                                     ;; get out of *Obstacle-Mode* if none of the successors
                                     ;; are obstacles.
                                 (PROGN
                                     (IF *DEBUG* (PROGN (PRINC 'Changing-to-Normal-Mode-now!)
                                                        (TERPRI) ))
                                     (SETF *Near-Obst-Edge* T)
                                     (SETQ *Obstacle-Mode* NIL) ) )
                             best-succ )
                      ;; else try the rest of sequence but remain in *Obstacle-Mode*.
                      (DO* ( (curr-seq-ls (REST search-sequence) (REST curr-seq-ls))
                             (curr-smode  (FIRST curr-seq-ls)  (FIRST curr-seq-ls))
                             (succ-list (get_succ_list curr-state curr-smode)
                                        (get_succ_list curr-state curr-smode) )
                             (best-succ (get_best_succ curr-state succ-list)
                                        (get_best_succ curr-state succ-list) ) )
                          ( (NOT (is_obstaclep (posn best-succ)))
                                (PROGN (IF *DEBUG* (PROGN (PRINC 'best-succ_is_ )
                                                          (PRINC best-succ) (TERPRI) ))
                                       best-succ) ) ) ) ) ) )


(DEFUN depth_threshold_reached (curr-depth)
       (COND ( (>= curr-depth  *zmapsize*) T)
             ( (<= curr-depth 1) T)
             ( T NIL) ) )


(DEFUN no_obstacles_in_succ_listp (state-list)
       (NOT (at_least_one_obstaclep state-list)) )


(DEFUN at_least_one_obstaclep (state-list)
       ;; return T if there is at least one posn in the state-list that is an obstacle;
       ;; else, NIL is returned.
       (COND ( (NULL state-list) NIL )
             ( (is_obstaclep (posn (FIRST state-list))) T )
             ( T (at_least_one_obstaclep (REST state-list)) ) ) )
```

```
(DEFUN get_succ_list (curr-state search-mode)
       ;; returns the list of successor states of the current state.
       (COND ((EQUAL search-mode 'fwd-level) (fwd_level_succ_list curr-state))
             ((EQUAL search-mode 'fwd-dive)  (fwd_dive_succ_list  curr-state))
             ((EQUAL search-mode 'fwd-rise)  (fwd_rise_succ_list  curr-state))

             ((EQUAL search-mode 'top-fwd-rl) (top_fwd_rl_succ_list curr-state))
             ((EQUAL search-mode 'bot-fwd-rl) (bot_fwd_rl_succ_list curr-state))
             ((EQUAL search-mode 'top-rl)     (top_rl_succ_list     curr-state))
             ((EQUAL search-mode 'bot-rl)     (bot_rl_succ_list     curr-state))
             ((EQUAL search-mode 'fwd-top)    (fwd_top_succ_list    curr-state))
             ((EQUAL search-mode 'fwd-bot)    (fwd_bot_succ_list    curr-state))
             ((EQUAL search-mode 'right-left) (right_left_succ_list curr-state))
             ((EQUAL search-mode 'back-up)    (back_up_succ_list    curr-state))

             ((EQUAL search-mode 'fwd-rise-and-level) (fwd_rise_and_level_succ_list curr-state))
             ((EQUAL search-mode 'fwd-dive-and-level) (fwd_dive_and_level_succ_list curr-state))
             ((EQUAL search-mode 'bot-all-and-rl) (bot_all_and_rl_succ_list curr-state))
             ((EQUAL search-mode 'top-all-and-rl) (top_all_and_rl_succ_list curr-state))
             ((EQUAL search-mode 'top-all) (top_all_succ_list curr-state))
             ((EQUAL search-mode 'bot-all) (bot_all_succ_list curr-state))
             ((EQUAL search-mode 'top)     (LIST (top_posn_state curr-state)))
             ((EQUAL search-mode 'bottom)  (LIST (bot_posn_state curr-state))) ) )


(DEFUN get_normal_mode_successor (curr-state succ-list)
       ;; returns the best successor state of the current state
       (LET ((best-succ (get_best_succ curr-state succ-list)))
            (COND ((is_obstaclep (posn best-succ))
                              (SETF *Obstacle-Mode* T) (successor curr-state) )
                  (T best-succ) ) ) )


(DEFUN path_mark (curr-state succ-state)
       (set_emap (posn curr-state) (+ (get_emap (posn curr-state))
                                      (local_cost (direction curr-state)
                                                  (posn curr-state)
                                                  (posn succ-state) ) )) )


(DEFUN det_search_mode (succ-state)
       (LET* ( (succ-posn  (posn succ-state))
               (succ-depth (z_coord succ-posn)) )
             (COND ((OR (is_obstaclep succ-posn) *Obstacle-Mode*)
                                                  (SETF *Obstacle-Mode* nil) )
                                                ;; (SETF *search-mode* '3D-all) )
                   ((> succ-depth *mission-depth*) (SETF *search-mode* 'fwd-rise))
                   ((= succ-depth *mission-depth*) (SETF *search-mode* 'fwd-level))
                   ((< succ-depth *mission-depth*) (SETF *search-mode* 'fwd-dive)) ) ) )


(DEFUN get_best_succ (curr-state succ-list)
       ;; returns the best successor (state) among those in succ-list
       (LET* ((best-succ (FIRST succ-list))
              (best-evalue (evaluation curr-state best-succ)) )
             (DO* ((rest-list (CDR succ-list) (CDR rest-list)))
                  ;; termination condition and result-form
                  ((NULL rest-list) best-succ)
                  ;; body of do loop
                  (LET* ((candidate-succ (FIRST rest-list))
                         (candidate-evalue (evaluation curr-state candidate-succ)) )
                        (COND ((< candidate-evalue best-evalue)
                               (SETF best-succ candidate-succ)
                               (SETF best-evalue candidate-evalue)) ) ) ) ) )
```

```
(DEFUN posn (state)
       ;; returns the positional coordinates of the state
       (SECOND state) )

(DEFUN direction (state)
       ;; returns the azimuth of the state
       (FIRST state) )

(DEFUN depth (state)
       (z_coord (posn state)) )


(DEFUN is_obstaclep (position)
       (IF (EQUAL (sense position) *infinity*)
           T
           NIL) )

(DEFUN not_obstaclep (position)
       (NOT (is_obstaclep position)) )

(DEFUN sense (posn)
       (AREF *emap* (x_coord posn) (y_coord posn) (z_coord posn)) )


(DEFUN make_vicinity_list (ref-posn)
       (LIST ref-posn
             (n_posn  ref-posn)
             (s_posn  ref-posn)
             (e_posn  ref-posn)
             (w_posn  ref-posn)
             (ne_posn ref-posn)
             (nw_posn ref-posn)
             (se_posn ref-posn)
             (sw_posn ref-posn) ) )


(DEFUN send_state_to_iris (state)
       (send_float (direction state))
       (send_posn_to_iris (posn state)) )

(DEFUN send_posn_to_iris (position)
       (send_float (x_coord position))
       (send_float (y_coord position))
       (send_float (z_coord position)) )


(DEFUN set_emap (coords value)
       (SETF (AREF *emap* (x_coord coords)
                          (y_coord coords)
                          (z_coord coords) ) value) )

(DEFUN get_emap (coords)
       (AREF *emap* (x_coord coords)
                    (y_coord coords)
                    (z_coord coords) ) )
```

```
;;; -*- Package: USE~ 'ode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;*******************************************************************************
;
;   Filename.......:    monitor.lisp
;   Author.........:    Ray Rogers
;   Modified by....:    Ong Seow Meng
;
;   Date Created...:    1989
;   Description....:    Contains all lisp code for generating the display on the
;                       Side Color Monitor.
;
;   Modifications..:
;
;*******************************************************************************

;;DEFINE VARIABLES

(DEFVAR *display-window*)
(DEFVAR *display-window-array*)
(DEFVAR *display-window-width*)
(DEFVAR *display-window-height*)
(DEFVAR *display-window-position*)
(DEFVAR *display-window-screen*)
(DEFVAR *display-window-pos*)

(DEFVAR *main-screen*)
(DEFVAR *screen-alu*)
(DEFVAR *start-alu*)
(DEFVAR *goal-alu*)
(DEFVAR *icon-alu*)
(DEFVAR *grid-alu*)
(DEFVAR *letter-alu*)
(DEFVAR *legend-box-alu*)

(DEFVAR *x-screen-org*)
(DEFVAR *y-screen-org*)
(DEFVAR *z-screen-org*)
(DEFVAR *scale*)
(DEFVAR *vert-scale*)
(DEFVAR xs)
(DEFVAR ys)
(DEFVAR xg)
(DEFVAR yg)
(DEFVAR xi)
(DEFVAR yi)
(DEFVAR *fixnum-dist-pu-coord* (TRUNCATE *real-horiz-dist-pu-coord*))
(DEFVAR *fixnum-vert-dist-pu-coord* (TRUNCATE *real-vert-dist-pu-coord*))


;;DEFINE WINDOW AND COLORS

(DEFFLAVOR my-color-flavor()
           (tv:window
            tv:graphics-mixin))

(DEFUN make-color-window
       (window-name position inside-width inside-height
        &rest options &key (superior (color:find-color-screen :create-p t))
        &allow-other-keys)
  (apply #'tv:make-window 'my-color-flavor
         :blinker-p nil
         :borders 2
         :save-bits t
         :expose-p t
         :label nil
         :name window-name
         :position position
         :inside-width inside-width
         :inside-height inside-height
         :superior superior
         options))
```

```
(DEFUN make-display-window ()
  (SETF *display-window*
        (make-color-window "Display-Window"
                           '(10 10) 1220 1000))
                          ;; '(50 50) 1150 850))
  (SETF *screen-alu* (SEND color:color-screen
                           :compute-color-alu
                           tv:alu-seta 0.3807 0.5125 1.0))
  (SEND *display-window* :set-erase-aluf *screen-alu*)
  (SEND *display-window* :refresh))


(DEFUN init-display ()
  (clear-scene)
  ;; actual poolsize is 1404 by 700 (or 20 X 10 auv-lengths)
  ;; i.e. approx a ratio of 2:1
  ;; Thus, we choose a screen-size of x-screen-size:y-screen-size = 2:1
  ;; The variable *scale* should be set at 1000/1400 = 500/700 = 0.7143)
  ;; Also, the vertical distance per unit coord is 10.0 and height of pool
  ;; is about 10 AUV heights,
  ;; Thus variable *vert-scale* should be set at 300/(10*10) = 3.0
  (SETF *x-screen-org*  100.)
  (SETF *y-screen-org*  55.)
  (SETF *z-screen-org* 650.)
  (SETF *scale* 0.7143)
  (SETF *vert-scale* 3.0)
  (draw_box)
  (draw_depth_box)
  'monitor-display-is-ready)


(DEFUN create-display-window()
  (SETF *main-screen* (SEND *terminal-io* :superior))
  (make-display-window)
  (SETF *display-window-pos*
        (SEND *display-window* :position))
  (SETF *display-window-screen*
        (SEND *display-window* :screen))
  (init-colors)
  'done-init-display-window)


(DEFUN clear-scene ()
  (tv:sheet-force-access (*display-window*)
    (SEND *display-window* :refresh)))


(DEFUN kill ()
  (SEND *display-window* :kill)
  'display-window-killed)


(DEFUN init-colors ()
  (SETF *start-alu* (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0.406 0.9535 0.2207))
  (SETF *goal-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 1.0 0.009008 0.8421))
  (SETF *path-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0.0 0.7 1.0))
  (SETF *obst-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0.5 0.5 0.5))
  (SETF *icon-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 1.0 0.0 0.2862))
  (SETF *grid-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0.9054 1.0 0.4847))
  (SETF *letter-alu* (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0 0 0))
  (SETF *legend-box-alu* (SEND *display-window-screen*
                               :compute-color-alu color:alu-x 0.745 0.7243 0.7976)))
```

```
(DEFUN draw_box ()
  (LET* ((x-screen-size 1000) (y-screen-size 550)
         (x-interval 50.) (y-interval 50.)
         (x-auv-lengths 20.) (y-auv-lengths 11.)
         (x-end-coord (+ *x-screen-org* x-screen-size))
         (y-end-coord (+ *y-screen-org* y-screen-size)) )
        (SEND *display-window* :draw-rectangle
              x-screen-size y-screen-size  *x-screen-org* *y-screen-org* *grid-alu*)
        ;;draw vertical lines
        (DO ((x-index *x-screen-org* (+ x-index x-interval)))
            ((> x-index x-end-coord) NIL)
            (SEND *display-window*
                  :draw-line x-index *y-screen-org* x-index y-end-coord *icon-alu*) )
        ;;draw horizontal lines
        (DO ((y-index *y-screen-org* (+ y-index y-interval)))
            ((> y-index y-end-coord) NIL)
            (SEND *display-window*
                  :draw-line *x-screen-org* y-index x-end-coord y-index *icon-alu*) ) ) )


(DEFUN draw_depth_box ()
  (LET* ((x-screen-size 1000) (z-screen-size 300)
         (x-interval 50.) (z-interval 30.)
         (x-auv-lengths 20.) (z-auv-height 10.)
         (x-end-coord (+ *x-screen-org* x-screen-size))
         (z-end-coord (+ *z-screen-org* z-screen-size)) )
        (SEND *display-window* :draw-rectangle
              x-screen-size z-screen-size  *x-screen-org* *z-screen-org* *grid-alu*)
        ;;draw vertical lines
        (DO ((x-index *x-screen-org* (+ x-index x-interval)))
            ((> x-index x-end-coord) NIL)
            (SEND *display-window*
                  :draw-line x-index *z-screen-org* x-index z-end-coord *icon-alu*) )
        ;;draw horizontal lines
        (DO ((z-index *z-screen-org* (+ z-index z-interval)))
            ((> z-index z-end-coord) NIL)
            (SEND *display-window*
                  :draw-line *x-screen-org* z-index x-end-coord z-index *icon-alu*) ) ) )


(DEFUN draw-icon (x y z)
  (SEND *display-window* :draw-filled-in-circle x y 6 *icon-alu*)
  (SEND *display-window* :draw-filled-in-circle x z 6 *icon-alu*) )


(DEFUN draw-start-pos (x y z)
  (SETF xs (+ (* x *scale*) *x-screen-org*))
  (SETF ys (+ (* y *scale*) *y-screen-org*))
  (SEND *display-window* :draw-filled-in-circle xs ys 20 *start-alu*)
  (SETF zs (+ (* z *vert-scale*) *z-screen-org*))
  (SEND *display-window* :draw-filled-in-circle xs zs 20 *start-alu*) )


(DEFUN draw-goal-pos (x y z)
  (SETF xg (+ (* x *scale*) *x-screen-org*))
  (SETF yg (+ (* y *scale*) *y-screen-org*))
  (SEND *display-window* :draw-filled-in-circle xg yg 20 *goal-alu*)
  (SETF zg (+ (* z *vert-scale*) *z-screen-org*))
  (SEND *display-window* :draw-filled-in-circle xg zg 20 *goal-alu*) )


(DEFUN draw-path-pos (x y z)
  ;; (x y z) are real position coordinates
  (SETF xp (+ (* x *scale*) *x-screen-org*))
  (SETF yp (+ (* y *scale*) *y-screen-org*))
  (SEND *display-window* :draw-filled-in-circle xp yp 12 *path-alu*)
  (SETF zp (+ (* z *vert-scale*) *z-screen-org*))
  (SEND *display-window* :draw-filled-in-circle xp zp 12 *path-alu*) )
```

```
(DEFUN draw_obst_pos (x y z)
   ;; (x y z) are real coordinates.
   (LET* ( (box-len (TRUNCATE (* *fixnum-dist-pu-coord* *scale*)))
           (half-box-len (TRUNCATE (/ box-len 2.)))
           (box-height (TRUNCATE (* *fixnum-vert-dist-pu-coord* *vert-scale*)))
           (half-box-height (TRUNCATE (/ box-height 2.)))
           (xobst     (TRUNCATE (+ (- (* x *scale*) half-box-len) *x-screen-org*)))
           (yobst     (TRUNCATE (+ (- (* y *scale*) half-box-len) *y-screen-org*)))
           (zobst     (TRUNCATE (+ (- (* z *vert-scale*) half-box-height) *z-screen-org*))) )
         (SEND *display-window* :draw-rectangle box-len box-len xobst yobst *obst-alu*)
         (SEND *display-window* :draw-rectangle box-len box-height xobst zobst *obst-alu*) ) )


(DEFUN move-icon (x y z)
  (setf xi (+ (* x *scale*) *x-screen-org*))
  (setf yi (+ (* y *scale*) *y-screen-org*))
  (SETF zi (+ (* z *vert-scale*) *z-screen-org*))
  (draw-icon xi yi zi) )


;;;main body
;;;prepare monitor

(create-display-window)
(init-display)
```

```
;;; -*- Package: USER; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;*************************************************************************************
;
;  Filename.......:  obstacle.lisp
;  Author.........:  Ong Seow Meng
;
;  Date Created...:  18 Aug 1989
;  Description....:  This file contains the lisp code for generating the obstacles used
;                    in the scenarios. A generic function called 'generate_random_obstacle'
;                    is defined for generating random obstacles; this function is also used
;                    for creating solid obstacles by specifying a value of 100% obstacle
;                    as the parameter. Code for 'obstacle growing', sending obstacle coords
;                    to iris for display, displaying obstacles on Side Color Monitor, etc.,
;                    is also included.
;
;  Notes..........:
;                   Structure of *ObstacleLs* is as follows:-
;                         ( (Obs-disposn C1 C2 ... Cn)  (Obs-disposn C1 C2 ... Cn) ..... )
;                    e.g., ( (long  (0 0 0) (1 0 0) (1 2 0)) (broad (1 1 1) (2 2 2)) .... )
;
;  Modifications..:
;
;
;*************************************************************************************

(DEFVAR *Obs01*)
(DEFVAR *Obs02*)
(DEFVAR *Obs03*)
(DEFVAR *Obs04*)
(DEFVAR *Obs05*)
(DEFVAR *Obs06*)
(DEFVAR *Obs07*)
(DEFVAR *Obs08*)
(DEFVAR *Obs11*)
(DEFVAR *Obs12*)
(DEFVAR *Obs13*)
(DEFVAR *Obs14*)
(DEFVAR *Obs15*)
(DEFVAR *Obs16*)
(DEFVAR *Obs21*)
(DEFVAR *Obs22*)
(DEFVAR *Obs23*)
(DEFVAR *Obs24*)
(DEFVAR *Obs25*)
(DEFVAR *Obs26*)
(DEFVAR *Obs31*)
(DEFVAR *Obs32*)
(DEFVAR *Obs33*)
(DEFVAR *Obs34*)
(DEFVAR *Obs35*)
(DEFVAR *Obs36*)


(DEFUN generate_random_obstacle (comment seed percent xorg yorg zorg xsize ysize zsize)
      (LET* ( (a 43411) (b 17) (c 640001) (x seed) (count 0)
              (Obst   (LIST comment)) )
            (DOTIMES (i xsize)
               (DOTIMES (j ysize)
                   (DOTIMES (k zsize)
                      (IF (< (/ (SETF x (MOD (+ (* a x) b) c)) c) (/ percent 100))
                          (PROGN (SETF obst
                                       (APPEND Obst (LIST (LIST (+ i xorg) (+ j yorg) (+ k zorg) ))
                                 (SETF count (1+ count)) ) ) ) ) ) )
            (TERPRI) (FORMAT T "Number of obstacle points = ") (PRINC count) (TERPRI)
            (LET ( (total-points (* xsize ysize zsize)) )
                 (FORMAT T "Total number of points = ") (PRINC total-points) (TERPRI)
                 (FORMAT T "Percentage obstacles = ") (PRINC (* (/ count total-points) 100.0)) )
            (LIST Obst) ) )
```

```
;; Obs01 is a wide wall obstacle in middle of nps pool.
(SETF *Obs01* (generate_random_obstacle 'wide_wall 10 100 2 9 4 8 1 3))

;; Obs02 is a high wall obstacle in the middle of nps pool.
(SETF *Obs02* (generate_random_obstacle 'high_wall 10 100 4 9 2 3 2 8))

;; Obs03 is a horizontal U-shaped obstacle (concave obstacle)
(SETF *Obs03* (LIST (APPEND (LIST 'horiz_U)
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3  8 3 1 5 7)))
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 8  8 3 1 5 7)))
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3 13 3 6 1 7))) )))

;; Obs04 is a vertical U-shaped obstacle (concave obstacle)
(SETF *Obs04* (LIST (APPEND (LIST 'horiz_U)
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3  8 3 1 5 7)))
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 8  8 3 1 5 7)))
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3 13 3 6 1 7))) )))

;; Obs05 is a tunnel (concave obstacle)
(SETF *Obs05* (LIST (APPEND (LIST 'small-tunnel)
                     ; the following is a vertical wall on the left of vehicle along y-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3  8 3 1 5 7)))
                     ; the following is a vertical wall on the right of vehicle along y-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 8  8 3 1 5 7)))
                     ; the following is a vertical wall at end of the tunnel along x-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3 13 3 6 1 7)))
                     ; the following is a top horizontal wall.
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3  8 3 6 6 1))))))

(SETF *Obs06* (LIST (APPEND (LIST 'medium-wide-tunnel)
                     ; the following is a vertical wall on the left of vehicle along y-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 2  8 3 1 5 7)))
                     ; the following is a vertical wall on the right of vehicle along y-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 8  8 3 1 5 7)))
                     ; the following is a vertical wall at end of the tunnel along x-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3 13 3 6 1 7)))
                     ; the following is a top horizontal wall.
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 3  8 3 6 6 1))))))

(SETF *Obs07* (LIST (APPEND (LIST 'very-wide-tunnel)
                     ; the following is a vertical wall on the left of vehicle along y-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 1  8 3 1 6 7)))
                     ; the following is a vertical wall on the right of vehicle along y-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 8  8 3 1 5 7)))
                     ; the following is a vertical wall at end of the tunnel along x-axis
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 2 13 3 7 1 7)))
                     ; the following is a top horizontal wall.
                     (REST (FIRST (generate_random_obstacle 'wall 10 100 2  8 3 7 6 1))))))

(SETF *Obs09* (generate_random_obstacle 'wall 10 100 3  8 3 6 6 1))

;; Obs1x series are random obstacles in a boxed region at location (2 7 3) and size 5 5 5.
(SETF *Obs11* (generate_random_obstacle 'random5  10  5 2 7 3 5 5 5))
(SETF *Obs12* (generate_random_obstacle 'random10 10 10 2 7 3 5 5 5))
(SETF *Obs13* (generate_random_obstacle 'random15 10 15 2 7 3 5 5 5))
(SETF *Obs14* (generate_random_obstacle 'random20 10 20 2 7 3 5 5 5))
(SETF *Obs15* (generate_random_obstacle 'random25 10 25 2 7 3 5 5 5))
(SETF *Obs16* (generate_random_obstacle 'random30 10 30 2 7 3 5 5 5))


;; Obs2x series are random obstacles in a boxed region at location (2 7 3) and size 6 3 5.
(SETF *Obs21* (generate_random_obstacle 'random5  20  5 2 7 3 6 3 5))
(SETF *Obs22* (generate_random_obstacle 'random10 20 10 2 7 3 6 3 5))
(SETF *Obs23* (generate_random_obstacle 'random15 20 15 2 7 3 6 3 5))
(SETF *Obs24* (generate_random_obstacle 'random20 20 20 2 7 3 6 3 5))
(SETF *Obs25* (generate_random_obstacle 'random25 20 25 2 7 3 6 3 5))
(SETF *Obs26* (generate_random_obstacle 'random30 20 30 2 7 3 6 3 5))
```

```
;; Obs3x series are random obstacles in a boxed region at location (1 8 1) and size 9 4 8.
;; This obstacle is spread across the width and height of the nps pool.
(SETF *Obs31* (generate_random_obstacle 'random5  31  5 1 8 1 9 4 8))
(SETF *Obs32* (generate_random_obstacle 'random10 31 10 1 8 1 9 4 8))
(SETF *Obs33* (generate_random_obstacle 'random15 31 15 1 8 1 9 4 8))
(SETF *Obs34* (generate_random_obstacle 'random20 31 20 1 8 1 9 4 8))
(SETF *Obs35* (generate_random_obstacle 'random25 31 25 1 8 1 9 4 8))
(SETF *Obs36* (generate_random_obstacle 'random30 31 30 1 8 1 9 4 8))


(DEFUN place_obs_ls ()
       (DO ((obst-ls *ObstacleLs* (REST obst-ls)))
            ((NULL obst-ls))
            ;; body of outer loop
            (LET* ((curr-obst (FIRST obst-ls))
                   (obst-disposn (FIRST curr-obst)))
                  (mapcar #'grow_obstacle (REST curr-obst)) ) )
       (display_obstacles_on_monitor) )


(DEFUN display_obstacles_on_monitor ()
       (DO ((obst-ls *ObstacleLs* (REST obst-ls)))
            ((NULL obst-ls))
            ;; body of outer loop
            (LET* ( (curr-obst (FIRST obst-ls)) )
                  (MAPCAR #'(LAMBDA (obst-posn)
                               (draw_obst_pos (* *real-horiz-dist-pu-coord* (y_coord obst-posn))
                                              (* *real-horiz-dist-pu-coord* (x_coord obst-posn))
                                              (* *real-vert-dist-pu-coord*  (z_coord obst-posn))) )
                          (REST curr-obst) ) ) ) )

(DEFUN send_obstacles_to_iris (obstacle)
       (MAPCAR #'send_posn_to_iris (REST obstacle)) )

(DEFUN grow_obstacle (coord)
       (mark_n   coord)
       (mark_tn  coord)
       (mark_bn  coord)
       (mark_s   coord)
       (mark_ts  coord)
       (mark_bs  coord)
       (mark_e   coord)
       (mark_te  coord)
       (mark_be  coord)
       (mark_w   coord)
       (mark_tw  coord)
       (mark_bw  coord)
       (mark_ne  coord)
       (mark_tne coord)
       (mark_bne coord)
       (mark_nw  coord)
       (mark_tnw coord)
       (mark_bnw coord)
       (mark_se  coord)
       (mark_tse coord)
       (mark_bse coord)
       (mark_sw  coord)
       (mark_tsw coord)
       (mark_bsw coord)
       (mark_top coord)
       (mark_bot coord) )


(DEFUN mark_n (coord)
       (set_emap (n_posn coord) *infinity*) )

(DEFUN mark_tn (coord)
       (set_emap (tn_posn coord) *infinity*) )
```

```
(DEFUN mark_bn (coord)
       (set_emap (bn_posn coord) *infinity*) )

(DEFUN mark_s (coord)
       (set_emap (s_posn coord) *infinity*) )

(DEFUN mark_ts (coord)
       (set_emap (ts_posn coord) *infinity*) )

(DEFUN mark_bs (coord)
       (set_emap (bs_posn coord) *infinity*) )

(DEFUN mark_e (coord)
       (set_emap (e_posn coord) *infinity*) )

(DEFUN mark_te (coord)
       (set_emap (te_posn coord) *infinity*) )

(DEFUN mark_be (coord)
       (set_emap (be_posn coord) *infinity*) )

(DEFUN mark_w (coord)
       (set_emap (w_posn coord) *infinity*) )

(DEFUN mark_tw (coord)
       (set_emap (tw_posn coord) *infinity*) )

(DEFUN mark_bw (coord)
       (set_emap (bw_posn coord) *infinity*) )

(DEFUN mark_ne (coord)
       (set_emap (ne_posn coord) *infinity*) )

(DEFUN mark_tne (coord)
       (set_emap (tne_posn coord) *infinity*) )

(DEFUN mark_bne (coord)
       (set_emap (bne_posn coord) *infinity*) )

(DEFUN mark_nw (coord)
       (set_emap (nw_posn coord) *infinity*) )

(DEFUN mark_tnw (coord)
       (set_emap (tnw_posn coord) *infinity*) )

(DEFUN mark_bnw (coord)
       (set_emap (bnw_posn coord) *infinity*) )

(DEFUN mark_se (coord)
       (set_emap (se_posn coord) *infinity*) )

(DEFUN mark_tse (coord)
       (set_emap (tse_posn coord) *infinity*) )

(DEFUN mark_bse (coord)
       (set_emap (bse_posn coord) *infinity*) )

(DEFUN mark_sw (coord)
       (set_emap (sw_posn coord) *infinity*) )

(DEFUN mark_tsw (coord)
       (set_emap (tsw_posn coord) *infinity*) )

(DEFUN mark_bsw (coord)
       (set_emap (bsw_posn coord) *infinity*) )

(DEFUN mark_top (coord)
       (set_emap (top_posn coord) *infinity*) )

(DEFUN mark_bot (coord)
       (set_emap (bot_posn coord) *infinity*) )
```

```
;;; -*- Package: USER; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;******************************************************************************************
;
;  Filename........:  posn.lisp
;  Author..........:  Ong Seow Meng
;
;  Date created....:  26 Aug 1989
;  Description.....:  This file contains the lisp code for generating the individual
;                     candidate successors.
;
;  Notes...........:  Pool oordinate system is as follows:
;;
;                              ^ Y axis (North)
;                             /
;                            /
;                origin .----> x axis (East)
;                       |
;                       |
;                       v Z axis
;
;
;  Modifications..:
;
;
;******************************************************************************************


(DEFUN n_posn (curr-posn)
       (list (x_coord curr-posn)
             (1+ (y_coord curr-posn))
             (z_coord curr-posn) ) )

(DEFUN n_posn_state (curr-state)
       (LET ((nposn (n_posn (posn curr-state))))
            (CONS 0 (LIST nposn)) ) )

(DEFUN s_posn (curr-posn)
       (list (x_coord curr-posn)
             (1- (y_coord curr-posn))
             (z_coord curr-posn) ) )

(DEFUN s_posn_state (curr-state)
       (LET ((sposn (s_posn (posn curr-state))))
            (CONS *PI* (LIST sposn)) ) )

(DEFUN e_posn (curr-posn)
       (cons (1+ (x_coord curr-posn)) (REST curr-posn)) )

(DEFUN e_posn_state (curr-state)
       (LET ((eposn (e_posn (posn curr-state))))
            (CONS *half-PI* (LIST eposn)) ) )

(DEFUN w_posn (curr-posn)
       (cons (1- (x_coord curr-posn)) (REST curr-posn)) )

(DEFUN w_posn_state (curr-state)
       (LET ((wposn (w_posn (posn curr-state))))
            (CONS (- *half-PI*) (LIST wposn)) ) )

(DEFUN ne_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (1+ (y_coord curr-posn))
             (z_coord curr-posn) ) )

(DEFUN ne_posn_state (curr-state)
       (LET ((neposn (ne_posn (posn curr-state))))
            (COND ( (= (direction curr-state) 0) (CONS *half-PI* (LIST neposn)))
                  ( T (CONS 0 (LIST neposn))) ) ) )
```

```
(DEFUN nw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (1+ (y_coord curr-posn))
             (z_coord curr-posn) ) )

(DEFUN nw_posn_state (curr-state)
       (LET ((nwposn (nw_posn (posn curr-state))))
            (COND ( (= (direction curr-state) 0) (CONS (- *half-PI*) (LIST nwposn)) )
                  ( T (CONS 0 (LIST nwposn))) ) ) )

(DEFUN se_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (1- (y_coord curr-posn))
             (z_coord curr-posn) ) )

(DEFUN se_posn_state (curr-state)
       (LET ((seposn (se_posn (posn curr-state))))
            (COND ( (= (direction curr-state) *half-PI*) (CONS  *PI* (LIST seposn)))
                  ( T (CONS *half-PI* (LIST seposn))) ) ) )

(DEFUN sw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (1- (y_coord curr-posn))
             (z_coord curr-posn) ) )

(DEFUN sw_posn_state (curr-state)
       (LET ((swposn (sw_posn (posn curr-state))))
            (COND ( (= (direction curr-state) *PI*) (CONS  (- *half-PI*) (LIST swposn)) )
                  ( T (CONS *PI* (LIST swposn))) ) ) )


;;;  top positions
;;;


(DEFUN tn_posn (curr-posn)
       (list (x_coord curr-posn)
             (1+ (y_coord curr-posn))
             (1- (z_coord curr-posn)) ) )

(DEFUN tn_posn_state (curr-state)
       (LET ((tnposn (tn_posn (posn curr-state))))
            (CONS 0 (LIST tnposn)) ) )

(DEFUN ts_posn (curr-posn)
       (list (x_coord curr-posn)
             (1- (y_coord curr-posn))
             (1- (z_coord curr-posn)) ) )

(DEFUN ts_posn_state (curr-state)
       (LET ((tsposn (ts_posn (posn curr-state))))
            (CONS *PI* (LIST tsposn)) ) )

(DEFUN tw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (y_coord curr-posn)
             (1- (z_coord curr-posn)) ) )

(DEFUN tw_posn_state (curr-state)
       (LET ((twposn (tw_posn (posn curr-state))))
            (CONS (- *half-PI*) (LIST twposn)) ) )

(DEFUN te_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (y_coord curr-posn)
             (1- (z_coord curr-posn)) ) )

(DEFUN te_posn_state (curr-state)
       (LET ((teposn (te_posn (posn curr-state))))
            (CONS *half-PI* (LIST teposn)) ) )
```

```
(DEFUN tne_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (1+ (y_coord curr-posn))
             (1- (z_coord curr-posn)) ) )

(DEFUN tne_posn_state (curr-state)
       (LET ((tneposn (tne_posn (posn curr-state))))
            (COND ( (= (direction curr-state) 0) (CONS *half-PI* (LIST tneposn)))
                  ( T (CONS 0 (LIST tneposn))) ) ) )

(DEFUN tnw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (1+ (y_coord curr-posn))
             (1- (z_coord curr-posn)) ) )

(DEFUN tnw_posn_state (curr-state)
       (LET ((tnwposn (tnw_posn (posn curr-state))))
            (COND ( (= (direction curr-state) 0) (CONS (- *half-PI*) (LIST tnwposn)) )
                  ( T (CONS 0 (LIST tnwposn))) ) ) )

(DEFUN tse_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (1- (y_coord curr-posn))
             (1- (z_coord curr-posn)) ) )

(DEFUN tse_posn_state (curr-state)
       (LET ((tseposn (tse_posn (posn curr-state))))
            (COND ( (= (direction curr-state) *half-PI*) (CONS  *PI* (LIST tseposn)))
                  ( T (CONS *half-PI* (LIST tseposn))) ) ) )

(DEFUN tsw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (1- (y_coord curr-posn))
             (1- (z_coord curr-posn)) ) )

(DEFUN tsw_posn_state (curr-state)
       (LET ((tswposn (tsw_posn (posn curr-state))))
            (COND ( (= (direction curr-state) *PI*) (CONS  (- *half-PI*) (LIST tswposn)) )
                  ( T (CONS *PI* (LIST tswposn))) ) ) )


;;;  Bottom positions
;;;

(DEFUN bn_posn (curr-posn)
       (list (x_coord curr-posn)
             (1+ (y_coord curr-posn))
             (1+ (z_coord curr-posn)) ) )

(DEFUN bn_posn_state (curr-state)
       (LET ((bnposn (bn_posn (posn curr-state))))
            (CONS 0 (LIST bnposn)) ) )

(DEFUN bs_posn (curr-posn)
       (list (x_coord curr-posn)
             (1- (y_coord curr-posn))
             (1+ (z_coord curr-posn)) ) )

(DEFUN bs_posn_state (curr-state)
       (LET ((bsposn (bs_posn (posn curr-state))))
            (CONS *PI* (LIST bsposn)) ) )

(DEFUN bw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (y_coord curr-posn)
             (1+ (z_coord curr-posn)) ) )

(DEFUN bw_posn_state (curr-state)
       (LET ((bwposn (bw_posn (posn curr-state))))
            (CONS (- *half-PI*) (LIST bwposn)) ) )
```

```
(DEFUN be_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (y_coord curr-posn)
             (1+ (z_coord curr-posn)) ) )

(DEFUN be_posn_state (curr-state)
       (LET ((beposn (be_posn (posn curr-state))))
            (CONS *half-PI* (LIST beposn)) ) )

(DEFUN bne_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (1+ (y_coord curr-posn))
             (1+ (z_coord curr-posn)) ) )

(DEFUN bne_posn_state (curr-state)
       (LET ((bneposn (bne_posn (posn curr-state))))
            (COND ( (= (direction curr-state) 0) (CONS *half-PI* (LIST bneposn)))
                  ( T (CONS 0 (LIST bneposn))) ) ) )

(DEFUN bnw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (1+ (y_coord curr-posn))
             (1+ (z_coord curr-posn)) ) )

(DEFUN bnw_posn_state (curr-state)
       (LET ((bnwposn (bnw_posn (posn curr-state))))
            (COND ( (= (direction curr-state) 0) (CONS (- *half-PI*) (LIST bnwposn)) )
                  ( T (CONS 0 (LIST bnwposn))) ) ) )

(DEFUN bse_posn (curr-posn)
       (list (1+ (x_coord curr-posn))
             (1- (y_coord curr-posn))
             (1+ (z_coord curr-posn)) ) )

(DEFUN bse_posn_state (curr-state)
       (LET ((bseposn (bse_posn (posn curr-state))))
            (COND ( (= (direction curr-state) *half-PI*) (CONS  *PI* (LIST bseposn)))
                  ( T (CONS *half-PI* (LIST bseposn))) ) ) )

(DEFUN bsw_posn (curr-posn)
       (list (1- (x_coord curr-posn))
             (1- (y_coord curr-posn))
             (1+ (z_coord curr-posn)) ) )

(DEFUN bsw_posn_state (curr-state)
       (LET ((bswposn (bsw_posn (posn curr-state))))
            (COND ( (= (direction curr-state) *PI*) (CONS  (- *half-PI*) (LIST bswposn)) )
                  ( T (CONS *PI* (LIST bswposn))) ) ) )

(DEFUN top_posn (curr-posn)
       (list (x_coord curr-posn)
             (y_coord curr-posn)
             (1- (z_coord curr-posn)) ) )

(DEFUN top_posn_state (curr-state)
       (LET ((topposn (top_posn (posn curr-state))))
            (CONS (direction curr-state) (LIST topposn)) ) )

(DEFUN bot_posn (curr-posn)
       (list (x_coord curr-posn)
             (y_coord curr-posn)
             (1+ (z_coord curr-posn)) ) )

(DEFUN bot_posn_state (curr-state)
       (LET ((botposn (bot_posn (posn curr-state))))
       (CONS (direction curr-state) (LIST botposn)) ) )
```

```
;;; -*- Package: USER; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;*************************************************************************************
;
;  Filename.......:  succ.lisp
;  Author.........:  Ong Seow Meng
;
;  Date created...:  21 Dec 1989
;  Description....:  This file contains the lisp functions that creates a list of
;                    candidate successors. This successor list is a function of the
;                    vehicle heading.
;
;  Modifications..:
;
;
;*************************************************************************************


(DEFUN fwd_dive_succ_list (curr-state)
       ;; returns list of forward candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (fwd_dive_succ0_list  curr-state))
                   ((rangep dir-q   3.5 4.5) (fwd_dive_succ4_list  curr-state))
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (fwd_dive_succ8_list  curr-state))
                   ((rangep dir-q -4.5 -3.5) (fwd_dive_succ12_list curr-state)) ) ) )

(DEFUN fwd_rise_succ_list (curr-state)
       ;; returns list of forward candidate successors above the current posn
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (fwd_rise_succ0_list  curr-state))
                   ((rangep dir-q   3.5 4.5) (fwd_rise_succ4_list  curr-state))
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (fwd_rise_succ8_list  curr-state))
                   ((rangep dir-q -4.5 -3.5) (fwd_rise_succ12_list curr-state)) ) ) )

(DEFUN fwd_level_succ_list (curr-state)
       ;; returns list of forward candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (fwd_level_succ0_list  curr-state))
                   ((rangep dir-q   3.5 4.5) (fwd_level_succ4_list  curr-state))
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (fwd_level_succ8_list  curr-state))
                   ((rangep dir-q -4.5 -3.5) (fwd_level_succ12_list curr-state)) ) ) )


(DEFUN fwd_rise_and_level_succ_list (curr-state)
       ;; returns list of forward rise and level candidate successors
       ;; in the x-y plane, in the current direction.
       (APPEND (fwd_rise_succ_list curr-state) (fwd_level_succ_list curr-state)) )


(DEFUN fwd_dive_and_level_succ_list (curr-state)
       ;; returns list of forward dive and level candidate successors
       ;; in the x-y plane, in the current direction.
       (APPEND (fwd_dive_succ_list curr-state) (fwd_level_succ_list curr-state)) )
```

```
(DEFUN top_fwd_rl_succ_list (curr-state)
      ;; returns list of top forward left and right candidate successors
      ;; in the x-y plane, in the current direction.
      (LET* ((curr-dir  (direction curr-state))
             (dir-q     (dir_quantum curr-dir)) )
            (COND ((rangep dir-q  -0.5 0.5) (LIST (tne_posn_state curr-state)
                                                  (tnw_posn_state curr-state) ) )
                  ((rangep dir-q   3.5 4.5) (LIST (tne_posn_state curr-state)
                                                  (tse_posn_state curr-state) ) )
                  ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                            (LIST (tse_posn_state curr-state)
                                                  (tsw_posn_state curr-state) ) )
                  ((rangep dir-q -4.5 -3.5) (LIST (tnw_posn_state curr-state)
                                                  (tsw_posn_state curr-state) ) ) ) ) )


(DEFUN bot_fwd_rl_succ_list (curr-state)
      ;; returns list of top forward left and right candidate successors
      ;; in the x-y plane, in the current direction.
      (LET* ((curr-dir  (direction curr-state))
             (dir-q     (dir_quantum curr-dir)) )
            (COND ((rangep dir-q  -0.5 0.5) (LIST (bne_posn_state curr-state)
                                                  (bnw_posn_state curr-state) ) )
                  ((rangep dir-q   3.5 4.5) (LIST (bne_posn_state curr-state)
                                                  (bse_posn_state curr-state) ) )
                  ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                            (LIST (bse_posn_state curr-state)
                                                  (bsw_posn_state curr-state) ) )
                  ((rangep dir-q -4.5 -3.5) (LIST (bnw_posn_state curr-state)
                                                  (bsw_posn_state curr-state) ) ) ) ) )


(DEFUN top_rl_succ_list (curr-state)
      ;; returns list of top left and right candidate successors
      ;; in the x-y plane, in the current direction.
      (LET* ((curr-dir  (direction curr-state))
             (dir-q     (dir_quantum curr-dir)) )
            (COND ((rangep dir-q  -0.5 0.5) (LIST (te_posn_state curr-state)
                                                  (tw_posn_state curr-state) ) )
                  ((rangep dir-q   3.5 4.5) (LIST (tn_posn_state curr-state)
                                                  (ts_posn_state curr-state) ) )
                  ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                            (LIST (te_posn_state curr-state)
                                                  (tw_posn_state curr-state) ) )
                  ((rangep dir-q -4.5 -3.5) (LIST (tn_posn_state curr-state)
                                                  (ts_posn_state curr-state) ) ) ) ) )


(DEFUN bot_rl_succ_list (curr-state)
      ;; returns list of bottom right and left candidate successors.
      ;; in the x-y plane, in the current direction.
      (LET* ((curr-dir  (direction curr-state))
             (dir-q     (dir_quantum curr-dir)) )
            (COND ((rangep dir-q  -0.5 0.5) (LIST (be_posn_state curr-state)
                                                  (bw_posn_state curr-state) ) )
                  ((rangep dir-q   3.5 4.5) (LIST_(bn_posn_state curr-state)
                                                  (bs_posn_state curr-state) ) )
                  ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                            (LIST (be_posn_state curr-state)
                                                  (bw_posn_state curr-state) ) )
                  ((rangep dir-q -4.5 -3.5) (LIST (bn_posn_state curr-state)
                                                  (bs_posn_state curr-state) ) ) ) ) )
```

```
(DEFUN right_left_succ_list (curr-state)
       ;; returns list of right and left candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (e_posn_state curr-state)
                                                   (w_posn_state curr-state) ) )
                   ((rangep dir-q   3.5 4.5) (LIST (n_posn_state curr-state)
                                                   (s_posn_state curr-state) ) )
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (e_posn_state curr-state)
                                                   (w_posn_state curr-state) ) )
                   ((rangep dir-q -4.5 -3.5) (LIST (n_posn_state curr-state)
                                                   (s_posn_state curr-state) ) ) ) ) )


(DEFUN top_all_succ_list (curr-state)
       ;; returns list of right and left candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (te_posn_state  curr-state)
                                                   (tw_posn_state  curr-state)
                                                   (top_posn_state curr-state) ) )
                   ((rangep dir-q   3.5 4.5) (LIST (tn_posn_state curr-state)
                                                   (ts_posn_state curr-state)
                                                   (top_posn_state curr-state) ) )
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (te_posn_state curr-state)
                                                   (tw_posn_state curr-state)
                                                   (top_posn_state curr-state) ) )
                   ((rangep dir-q -4.5 -3.5) (LIST (tn_posn_state curr-state)
                                                   (ts_posn_state curr-state)
                                                   (top_posn_state curr-state) ) ) ) ) )


(DEFUN bot_all_succ_list (curr-state)
       ;; returns list of right and left candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (be_posn_state  curr-state)
                                                   (bw_posn_state  curr-state)
                                                   (bot_posn_state curr-state) ) )
                   ((rangep dir-q   3.5 4.5) (LIST (bn_posn_state curr-state)
                                                   (bs_posn_state curr-state)
                                                   (bot_posn_state curr-state) ) )
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (be_posn_state curr-state)
                                                   (bw_posn_state curr-state)
                                                   (bot_posn_state curr-state) ) )
                   ((rangep dir-q -4.5 -3.5) (LIST (bn_posn_state curr-state)
                                                   (bs_posn_state curr-state)
                                                   (bot_posn_state curr-state) ) ) ) ) )
```

```
(DEFUN top_all_and_rl_succ_list (curr-state)
       ;; returns list of right and left candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (e_posn_state   curr-state)
                                                   (w_posn_state   curr-state)
                                                   (te_posn_state  curr-state)
                                                   (tw_posn_state  curr-state)
                                                   (top_posn_state curr-state) ) )
                   ((rangep dir-q   3.5 4.5) (LIST (n_posn_state curr-state)
                                                   (s_posn_state curr-state)
                                                   (tn_posn_state curr-state)
                                                   (ts_posn_state curr-state)
                                                   (top_posn_state curr-state) ) )
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (e_posn_state curr-state)
                                                   (w_posn_state curr-state)
                                                   (te_posn_state curr-state)
                                                   (tw_posn_state curr-state)
                                                   (top_posn_state curr-state) ) )
                   ((rangep dir-q -4.5 -3.5) (LIST (n_posn_state curr-state)
                                                   (s_posn_state curr-state)
                                                   (tn_posn_state curr-state)
                                                   (ts_posn_state curr-state)
                                                   (top_posn_state curr-state) ) ) ) ) )


(DEFUN bot_all_and_rl_succ_list (curr-state)
       ;; returns list of right and left candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (e_posn_state   curr-state)
                                                   (w_posn_state   curr-state)
                                                   (be_posn_state  curr-state)
                                                   (bw_posn_state  curr-state)
                                                   (bot_posn_state curr-state) ) )
                   ((rangep dir-q   3.5 4.5) (LIST (n_posn_state curr-state)
                                                   (s_posn_state curr-state)
                                                   (bn_posn_state curr-state)
                                                   (bs_posn_state curr-state)
                                                   (bot_posn_state curr-state) ) )
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (e_posn_state curr-state)
                                                   (w_posn_state curr-state)
                                                   (be_posn_state curr-state)
                                                   (bw_posn_state curr-state)
                                                   (bot_posn_state curr-state) ) )
                   ((rangep dir-q -4.5 -3.5) (LIST (n_posn_state curr-state)
                                                   (s_posn_state curr-state)
                                                   (bn_posn_state curr-state)
                                                   (bs_posn_state curr-state)
                                                   (bot_posn_state curr-state) ) ) ) ) )


(DEFUN back_up_succ_list (curr-state)
       ;; returns list of all candidate successors behind the curr-state,
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (back_up_succ0_list  curr-state))
                   ((rangep dir-q   3.5 4.5) (back_up_succ4_list  curr-state))
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (back_up_succ8_list  curr-state))
                   ((rangep dir-q -4.5 -3.5) (back_up_succ12_list curr-state)) ) ) )
```

```
(DEFUN fwd_top_succ_list (curr-state)
       ;; returns the one and only forward top candidate successors
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (tn_posn_state curr-state)))
                   ((rangep dir-q   3.5 4.5) (LIST (te_posn_state curr-state)))
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (ts_posn_state curr-state)) )
                   ((rangep dir-q -4.5 -3.5) (LIST (tw_posn_state curr-state))) ) ) )

(DEFUN fwd_bot_succ_list (curr-state)
       ;; returns the one and only forward bottom candidate successor
       ;; in the x-y plane, in the current direction.
       (LET* ((curr-dir  (direction curr-state))
              (dir-q     (dir_quantum curr-dir)) )
             (COND ((rangep dir-q  -0.5 0.5) (LIST (bn_posn_state curr-state)))
                   ((rangep dir-q   3.5 4.5) (LIST (be_posn_state curr-state)))
                   ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (LIST (bs_posn_state curr-state)) )
                   ((rangep dir-q -4.5 -3.5) (LIST (bw_posn_state curr-state))) ) ) )


(DEFUN fwd_dive_succ0_list (curr-state)
       (LIST (bn_posn_state  curr-state)
             (bne_posn_state curr-state)
             (bnw_posn_state curr-state) ) )

(DEFUN fwd_dive_succ4_list (curr-state)
       (LIST (bne_posn_state curr-state)
             (be_posn_state  curr-state)
             (bse_posn_state curr-state) ) )

(DEFUN fwd_dive_succ8_list (curr-state)
       (LIST (bse_posn_state curr-state)
             (bs_posn_state  curr-state)
             (bsw_posn_state curr-state) ) )

(DEFUN fwd_dive_succ12_list (curr-state)
       (LIST (bsw_posn_state curr-state)
             (bw_posn_state  curr-state)
             (bnw_posn_state curr-state) ) )


(DEFUN fwd_rise_succ0_list (curr-state)
       (LIST (tn_posn_state  curr-state)
             (tne_posn_state curr-state)
             (tnw_posn_state curr-state) ) )

(DEFUN fwd_rise_succ4_list (curr-state)
       (LIST (tne_posn_state curr-state)
             (te_posn_state  curr-state)
             (tse_posn_state curr-state) ) )

(DEFUN fwd_rise_succ8_list (curr-state)
       (LIST (tse_posn_state curr-state)
             (ts_posn_state  curr-state)
             (tsw_posn_state curr-state) ) )

(DEFUN fwd_rise_succ12_list (curr-state)
       (LIST (tsw_posn_state curr-state)
             (tw_posn_state  curr-state)
             (tnw_posn_state curr-state) ) )

(DEFUN fwd_level_succ0_list (curr-state)
       (LIST (n_posn_state  curr-state)
             (ne_posn_state curr-state)
             (nw_posn_state curr-state) ) )
```

```
(DEFUN fwd_level_succ4_list (curr-state)
       (LIST (ne_posn_state curr-state)
             (e_posn_state  curr-state)
             (se_posn_state curr-state) ) )


(DEFUN fwd_level_succ8_list (curr-state)
       (LIST (se_posn_state curr-state)
             (s_posn_state  curr-state)
             (sw_posn_state curr-state) ) )


(DEFUN fwd_level_succ12_list (curr-state)
       (LIST (sw_posn_state curr-state)
             (w_posn_state  curr-state)
             (nw_posn_state curr-state) ) )


(DEFUN back_up_succ0_list (curr-state)
       (LIST (s_posn_state   curr-state)
             (se_posn_state  curr-state)
             (sw_posn_state  curr-state)
             (ts_posn_state  curr-state)
             (tse_posn_state curr-state)
             (tsw_posn_state curr-state)
             (bs_posn_state  curr-state)
             (bse_posn_state curr-state)
             (bsw_posn_state curr-state) ) )

(DEFUN back_up_succ4_list (curr-state)
       (LIST (nw_posn_state  curr-state)
             (sw_posn_state  curr-state)
             (w_posn_state   curr-state)
             (tnw_posn_state curr-state)
             (tsw_posn_state curr-state)
             (tw_posn_state  curr-state)
             (bnw_posn_state curr-state)
             (bsw_posn_state curr-state)
             (bw_posn_state  curr-state) ) )

(DEFUN back_up_succ8_list (curr-state)
       (LIST (n_posn_state   curr-state)
             (ne_posn_state  curr-state)
             (nw_posn_state  curr-state)
             (tn_posn_state  curr-state)
             (tne_posn_state curr-state)
             (tnw_posn_state curr-state)
             (bn_posn_state  curr-state)
             (bne_posn_state curr-state)
             (bnw_posn_state curr-state) ) )


(DEFUN back_up_succ12_list (curr-state)
       (LIST (ne_posn_state  curr-state)
             (se_posn_state  curr-state)
             (e_posn_state   curr-state)
             (tne_posn_state curr-state)
             (tse_posn_state curr-state)
             (te_posn_state  curr-state)
             (bne_posn_state curr-state)
             (bse_posn_state curr-state)
             (be_posn_state  curr-state) ) )
```

```
(DEFUN 3D_fwd_succ_list (curr-state)
      ;; returns list of all forward candidate
      ;; successors in the current direction.
      (LET* ((curr-dir  (direction curr-state))
             (dir-q      (dir_quantum curr-dir)) )
            (COND ((rangep dir-q  -0.5 0.5)  (3D_succ0_list  curr-state))
                  ((rangep dir-q   3.5 4.5)  (3D_succ4_list  curr-state))
                  ((OR (rangep dir-q -8.1 -7.5) (rangep dir-q 7.5 8.1))
                                             (3D_succ8_list  curr-state))
                  ((rangep dir-q -4.5 -3.5)  (3D_succ12_list curr-state)) ) ) )


(DEFUN 3D_succ0_list (curr-state)
      (LIST (n_posn_state   curr-state)
            (ne_posn_state  curr-state)
            (nw_posn_state  curr-state)
            (tn_posn_state  curr-state)
            (tne_posn_state curr-state)
            (tnw_posn_state curr-state)
            (bn_posn_state  curr-state)
            (bne_posn_state curr-state)
            (bnw_posn_state curr-state) ) )


(DEFUN 3D_succ4_list (curr-state)
      (LIST (ne_posn_state  curr-state)
            (e_posn_state   curr-state)
            (se_posn_state  curr-state)
            (tne_posn_state curr-state)
            (te_posn_state  curr-state)
            (tse_posn_state curr-state)
            (bne_posn_state curr-state)
            (be_posn_state  curr-state)
            (bse_posn_state curr-state) ) )


(DEFUN 3D_succ8_list (curr-state)
      (LIST (se_posn_state  curr-state)
            (s_posn_state   curr-state)
            (sw_posn_state  curr-state)
            (tse_posn_state curr-state)
            (ts_posn_state  curr-state)
            (tsw_posn_state curr-state)
            (bse_posn_state curr-state)
            (bs_posn_state  curr-state)
            (bsw_posn_state curr-state) ) )


(DEFUN 3D_succ12_list (curr-state)
      (LIST (sw_posn_state  curr-state)
            (w_posn_state   curr-state)
            (nw_posn_state  curr-state)
            (tsw_posn_state curr-state)
            (tw_posn_state  curr-state)
            (tnw_posn_state curr-state)
            (bsw_posn_state curr-state)
            (bw_posn_state  curr-state)
            (bnw_posn_state curr-state) ) )
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER -*-

;*****************************************************************************************
;
;  Filename........:  sym-iris-comms.lisp
;
;  Modifications:
;          26 Feb 90    1. Changed the following port numbers due to IRIS OS Upgrade:
;                            *remote-port1*  1052
;                            *remote-port2*  1051
;
;
;  "Talk" is an object to send and to receive data across a network.
;
;  usage : (send talk :init-destination-host 'iris2)  ; get remote host object
;          (send talk :start-iris)                     ; make connection
;          (send talk :put-iris data)                  ; send data
;          (send talk :get-iris)                       ; get data from remote host
;          (send talk :stop-iris)                      ; close communication
;          (send talk :reuse-iris)                     ; open closed communication
;          (send talk :change-iris-ports)              ; switch from iris2 full-duplex
;                                                      ; comms to iris5 semi-duplex
;
;*****************************************************************************************


(defvar talk)


;
; library functions to be used by flavor conversation-with-iris.
;


(defmacro loopfor (var init test expl)
  `(prog ()
         (setq ,var ,init)
         tag
            ,expl
            (setq ,var (1+ ,var))
            (if (= ,var ,test) (return t) (go tag))))


(defun convert-number-to-string (n)
  (princ-to-string n))

(defun convert-string-to-integer (str &optional (radix 10))
  (do ((j 0 (+ j 1))
       (n 0 (+ (* n radix) (digit-char-p (char str j) radix))))
      ((= j (length str)) n)))

(defun find-period-index (str)
  (catch 'exit
    (dotimes (x (length str) nil)
      (if (equal (char str x) (char "." 0))
          (throw 'exit x)))))

(defun get-leftside-of-real (str &optional (radix 10))
  (do  ((j 0 (1+ j))
        (n 0 (+ (* n radix) (digit-char-p (char str j) radix))))
       ((or (null (digit-char-p (char str j) radix)) (= j (length str)))  n)))

(defun get-rightside-of-real (str &optional (radix 10))
  (do ((index (1+ (find-period-index str)) (1+ index))
       (factor 0.10 (* factor 0.10))
       (n 0.0 (+ n (* factor (digit-char-p (char str index) radix)))))
      ((= index (length str)) n )))

(defun convert-string-to-real (str &optional (radix 10))
  (+ (float (get-leftside-of-real str radix)) (get-rightside-of-real str radix)))
```

```
;
; port number definitions: Iris2 uses full duplex comms so ports are set up for
;      this default.  Iris5 uses semiduplex comms (the same port for send and
;      receive) and will have both ports set to *remote-port1*.
;


;; The following port numbers, *remote-port1* and *remote-port2*
;; has been changed due to IRIS OS upgrade.
;;(defvar *remote-port1* 1027)                    ; this is the remote send port
;;(defvar *remote-port2* 1026)                    ; this is the remote receive port
(defvar *remote-port1* 1052)                    ; this is the remote send port
(defvar *remote-port2* 1051)                    ; this is the remote receive port

(defvar *local-talk-port* 1500)                 ; this is the local send port
(defvar *local-listen-port* 1501)               ; this is the local receive port

(SETF *remote-port1*      1052)                   ; this is the remote send port
(SETF *remote-port2*      1051)                   ; this is the remote receive port
(SETF *local-talk-port*   1500)                 ; this is the local send port
(SETF *local-listen-port* 1501)               ; this is the local receive port




;
;  conversation-with-iris flavor definition
;
;
;  This definition is not restricted to iris, but it can be
;  used with any host as long as the remote host does not
;  already use ports 1027 or 1026 for its own purposes.
;


(defflavor conversation-with-iris ((talking-port-number      *remote-port1*)
                                   (listening-port-number    *remote-port2*)
                                   (local-talk-port-number   *local-talk-port*)
                                   (local-listen-port-number *local-listen-port*)
                                   (talking-stream)
                                   (listening-stream)
                                   (destination-host-object)
                                  )
                                  ()
                                  :initable-instance-variables)


(defmethod (:init-destination-host conversation-with-iris)
          (name-of-host)
  (setf destination-host-object (net:parse-host name-of-host)))


(defmethod (:change-iris-ports conversation-with-iris)
          ()
  (setf talking-port-number    *remote-port1*)      ;sets iris5 semi-duplex comm ports.
  (setf listening-port-number  *remote-port1*))


(defmethod (:start-iris conversation-with-iris)
          ()
  (setf talking-stream
        (tcp:open-tcp-stream destination-host-object
                             talking-port-number
                             local-talk-port-number))
  (setf listening-stream
        (tcp:open-tcp-stream destination-host-object
                             listening-port-number
                             local-listen-port-number))
  (terpri)
  (princ "A conversation with the iris machine has been initiated.")
  (terpri))
```

```
(defmethod (:reuse-iris conversation-with-iris)
           ()
  (send self :start-iris))


(defun read-string (stream num-chars)
  (let ((out-string ""))
    (dotimes (i num-chars)
      (setf out-string (string-append out-string (read-char stream))))
    out-string))


(defmethod (:get-iris conversation-with-iris)
           ()
  (let* ((typebuffer   " ")
         (lengthbuffer "    ")
         (buffer       " ")
         (buffer-length 1))
    (progn
      (setf typebuffer
            (read-string listening-stream 1))
      (setf lengthbuffer
            (read-string listening-stream 4))
      (setf buffer-length
            (convert-string-to-integer lengthbuffer))
      (setf buffer
            (read-string listening-stream buffer-length))


        (cond ((equal typebuffer "I") (convert-string-to-integer buffer))
              ((equal typebuffer "R") (convert-string-to-real    buffer))
              ((equal typebuffer "C") buffer)
              (t nil)))))


(defvar *step-var* 0)


(defun my-write-string(string stream)
  (let* ((num-chars (length string)))
    (dotimes (i num-chars)
      (write-char (aref string i) stream))))


(defmethod (:put-iris conversation-with-iris)
           (object)

  (let* ((buffer (cond
                  ((equal (type-of object) 'bignum) (convert-number-to-string object))
                  ((equal (type-of object) 'fixnum) (convert-number-to-string object))
                  ((equal (type-of object) 'single-float) (convert-number-to-string object))
                  ((equal (type-of object) 'string) object)
                  (t "error")))

         (buffer-length  (length buffer))

         (typebuffer     (cond ((equal (type-of object) 'bignum) "I")
                               ((equal (type-of object) 'fixnum) "I")
                               ((equal (type-of object) 'single-float) "R")
                               ((equal (type-of object) 'string) "C")
                               (t "C")))

         (lengthbuffer   (convert-number-to-string buffer-length)))
    (progn
      (my-write-string typebuffer talking-stream)
      (send talking-stream :force-output)


        (if (= (length lengthbuffer) 4)
            (write-string lengthbuffer talking-stream)
            (progn
               (loopfor *step-var* (length lengthbuffer) 4
                          (write-string "0" talking-stream))

               (my-write-string lengthbuffer talking-stream)
               ))

        (send talking-stream :force-output)


        (my-write-string buffer talking-stream)
        (send talking-stream :force-output)

        )))



(defmethod (:stop-iris conversation-with-iris)
           ()
  (progn (send listening-stream :close)
         (send talking-stream :close))
  (terpri)
  (princ "A conversation with the iris machine has been closed.")
  (terpri))


(setf talk (make-instance 'conversation-with-iris))


(defun choose-iris (*host-name*)                        ;use this function when selecting comms
  (cond                                                 ;from the keyboard
    ((equal *host-name* 'iris2)
     (setq *host-name* 'iris2)
     (send talk :init-destination-host *host-name*)      ;use iris2 as default output.
     (terpri)
     (princ "Iris2 communications selected.")
     (terpri))
    ((equal *host-name* 'iris5)
     (setq *host-name* 'iris5)
;     (send talk :change-iris-ports)                    ;select semi-duplex comm ports.
     (send talk :init-destination-host *host-name*)
     (terpri)
     (princ "Iris5 communications selected.")
     (terpri))))


(defun select-iris2()                                   ;use these two functions when using
       (setf *host-name* 'iris2)                        ;the mouse-driven control panel
       (send talk :init-destination-host *host-name*)
       (terpri)
       (princ "Iris2 communications selected from Control Panel.")
       (terpri))

(defun select-iris5()
       (setf *host-name* 'iris5)
       (send talk :init-destination-host *host-name*)
       (terpri)
       (princ "Iris5 communications selected from Control Panel.")
       (terpri))
```

```
(defun start-con()
  (send talk :start-iris))

(defun get_data()
  (send talk :get-iris))

(defun send_float(single-float)
  (send talk :put-iris single-float))

(defun send_string(string)
  (send talk :put-iris string))

(defun end-con()
  (send talk :stop-iris))

(defun restart()
  (send talk :reuse-iris))
```

```
;;; -*- Package: KEE; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;*****************************************************************************
;
;  Filename......:  missions.lisp
;  Author........:  Ong Seow Meng
;
;  Date Created..:  20 Jan 90
;  Description...:  Contains the methods referenced by the following UNITS in
;                   the MPES knowledge base.
;                   [UNIT]=[MISSIONS]
;                   [UNIT]=[TRANSIT.POOL]
;                   [UNIT]=[NPS.POOL]
;                   [UNIT]=[OPS.ORDERS]
;
;  Modifications.:
;
;
;
;*****************************************************************************


;=============================================================================
; NPS.POOL UNIT methods
;=============================================================================


  ;---------------------------------------------------------------------------
  ; INIT_OBSTACLES method is for [unit::slot]=[NPS.POOL::init-obstacles]
  ;---------------------------------------------------------------------------
  (DEFUN init_obstacles (THISUNIT)
         ;; This method is to be activated OFFLINE and only ONCE to set the values.
         (REMOVE.ALL.VALUES 'nps.pool 'obs01)
         (REMOVE.ALL.VALUES 'nps.pool 'obs02)
         (REMOVE.ALL.VALUES 'nps.pool 'obs03)
         (REMOVE.ALL.VALUES 'nps.pool 'obs04)
         (REMOVE.ALL.VALUES 'nps.pool 'obs05)
         (REMOVE.ALL.VALUES 'nps.pool 'obs06)
         (REMOVE.ALL.VALUES 'nps.pool 'obs07)
      ;;  (REMOVE.ALL.VALUES 'nps.pool 'obs08)
         (REMOVE.ALL.VALUES 'nps.pool 'obs11)
         (REMOVE.ALL.VALUES 'nps.pool 'obs12)
         (REMOVE.ALL.VALUES 'nps.pool 'obs13)
         (REMOVE.ALL.VALUES 'nps.pool 'obs14)
         (REMOVE.ALL.VALUES 'nps.pool 'obs15)
         (REMOVE.ALL.VALUES 'nps.pool 'obs16)
         (REMOVE.ALL.VALUES 'nps.pool 'obs21)
         (REMOVE.ALL.VALUES 'nps.pool 'obs22)
         (REMOVE.ALL.VALUES 'nps.pool 'obs23)
         (REMOVE.ALL.VALUES 'nps.pool 'obs24)
         (REMOVE.ALL.VALUES 'nps.pool 'obs25)
         (REMOVE.ALL.VALUES 'nps.pool 'obs26)
         (REMOVE.ALL.VALUES 'nps.pool 'obs31)
         (REMOVE.ALL.VALUES 'nps.pool 'obs32)
         (REMOVE.ALL.VALUES 'nps.pool 'obs33)
         (REMOVE.ALL.VALUES 'nps.pool 'obs34)
         (REMOVE.ALL.VALUES 'nps.pool 'obs35)
         (REMOVE.ALL.VALUES 'nps.pool 'obs36)

         (PUT.VALUE 'nps.pool 'obs01 USER::*Obs01*)
         (PUT.VALUE 'nps.pool 'obs02 USER::*Obs02*)
         (PUT.VALUE 'nps.pool 'obs03 USER::*Obs03*)
         (PUT.VALUE 'nps.pool 'obs04 USER::*Obs04*)
         (PUT.VALUE 'nps.pool 'obs05 USER::*Obs05*)
         (PUT.VALUE 'nps.pool 'obs06 USER::*Obs06*)
         (PUT.VALUE 'nps.pool 'obs07 USER::*Obs07*)
      ;;  (PUT.VALUE 'nps.pool 'obs08 USER::*Obs08*)
         (PUT.VALUE 'nps.pool 'obs11 USER::*Obs11*)
         (PUT.VALUE 'nps.pool 'obs12 USER::*Obs12*)
         (PUT.VALUE 'nps.pool 'obs13 USER::*Obs13*)
         (PUT.VALUE 'nps.pool 'obs14 USER::*Obs14*)
```

```
        (PUT.VALUE 'nps.pool 'obs15 USER::*Obs15*)
        (PUT.VALUE 'nps.pool 'obs16 USER::*Obs16*)
        (PUT.VALUE 'nps.pool 'obs21 USER::*Obs21*)
        (PUT.VALUE 'nps.pool 'obs22 USER::*Obs22*)
        (PUT.VALUE 'nps.pool 'obs23 USER::*Obs23*)
        (PUT.VALUE 'nps.pool 'obs24 USER::*Obs24*)
        (PUT.VALUE 'nps.pool 'obs25 USER::*Obs25*)
        (PUT.VALUE 'nps.pool 'obs26 USER::*Obs26*)
        (PUT.VALUE 'nps.pool 'obs31 USER::*Obs31*)
        (PUT.VALUE 'nps.pool 'obs32 USER::*Obs32*)
        (PUT.VALUE 'nps.pool 'obs33 USER::*Obs33*)
        (PUT.VALUE 'nps.pool 'obs34 USER::*Obs34*)
        (PUT.VALUE 'nps.pool 'obs35 USER::*Obs35*)
        (PUT.VALUE 'nps.pool 'obs36 USER::*Obs36*) )


;==============================================================================
; PANELS UNIT methods
;==============================================================================

 ;-----------------------------------------------------------------------------
 ; RESET_SCREEN method is for [unit::slot]=[PANELS::reset-screen]
 ;              -- called by [unit::slot]=[EXECUTOR::abort-mission].
 ;-----------------------------------------------------------------------------
 (DEFUN reset_screen (THISUNIT)
        (UNITMSG 'viewport-auv.status.panel.2       'close-panel!)
        (UNITMSG 'viewport-mission.status.panel.13 'close-panel!)
        (UNITMSG 'viewport-execute.abort.panel.16  'close-panel!)
        (UNITMSG 'enter-parameters-prompt          'close!)
        (UNITMSG 'viewport-transit.pool.1          'close-panel!)
        (UNITMSG 'viewport-user.prompt.panel.3     'open-panel!)
        (UNITMSG 'select-mission-prompt            'open!)
        (UNITMSG 'viewport-select.mission.panel.5  'open-panel!) )


;==============================================================================
; TRANSIT.POOL UNIT methods
;==============================================================================

 ;-----------------------------------------------------------------------------
 ; SELECT_TRANSIT_POOL method is for [unit::slot]=[TRANSIT_POOL::select-mission]
 ;                       The active-image TRANSIT POOL in SELECT.MISSION.PANEL is
 ;                       attached to this method.
 ;-----------------------------------------------------------------------------
 (DEFUN select_transit_pool (THISUNIT)
        (UNITMSG 'viewport-select.mission.panel.5 'close-panel!)
        (UNITMSG 'select-mission-prompt 'close!)
        (UNITMSG 'enter-parameters-prompt 'open!)
        (UNITMSG 'viewport-transit.pool.1 'open-panel!) )

 ;-----------------------------------------------------------------------------
 ; INITIATE_TRANSIT_POOL method is for [unit::slot]=[TRANSIT_POOL::initiate-mission]
 ;                         The active-image titled "OK" in TRANSIT.POOL image-panel is
 ;                         attached to this method.
 ;-----------------------------------------------------------------------------
 (DEFUN initiate_transit_pool (THISUNIT)
        (SETF USER::*DEBUG* NIL)
       ;; (future enhancement) check all entries are valid and satisfy cardinality
       ;; constraints before writing mission orders.
        (UNITMSG THISUNIT 'write-mission.orders)
        (UNITMSG 'viewport-user.prompt.panel.3 'close-panel!)
        (REMOVE.ALL.VALUES 'mission.status.panel 'mission-status)
        (UNITMSG 'viewport-mission.status.panel.13 'open-panel!)
        (clear.unstructured.facts)
        (ASSERT '(TEXT 'planning-phase) 'mission.planning.controller) )
```

```
;===============================================================================
; MISSIONS UNIT methods
;===============================================================================

  ;-----------------------------------------------------------------------------
  ; INIT_DATA_SLOTS is for  [unit::slot]=[<MISSIONS>::init-data-slots]
  ;-----------------------------------------------------------------------------
  (DEFUN init_data_slots (THISUNIT)
        (IF USER::*DEBUG* (FORMAT T "~%   Entered function 'init_data_slots'."))
        (REMOVE.ALL.VALUES THISUNIT 'area-operation)
        (REMOVE.ALL.VALUES THISUNIT 'goal-posn)
        (REMOVE.ALL.VALUES THISUNIT 'hovering-mode)
        (REMOVE.ALL.VALUES THISUNIT 'initial-heading)
        (REMOVE.ALL.VALUES THISUNIT 'mission-depth)
        (REMOVE.ALL.VALUES THISUNIT 'mission-speed)
        (REMOVE.ALL.VALUES THISUNIT 'safety-radius)
        (REMOVE.ALL.VALUES THISUNIT 'start-posn)
        (REMOVE.ALL.VALUES THISUNIT 'threat)
        (REMOVE.ALL.VALUES THISUNIT 'time-available)
        (IF USER::*DEBUG* (FORMAT T "~%   Exit function 'init_data_slots'.")) )

  ;-----------------------------------------------------------------------------
  ; WRITE_MISSION.ORDERS is for  [unit::slot]=[<MISSIONS>::write-mission.orders]
  ;-----------------------------------------------------------------------------
  (DEFUN write_mission.orders (THISUNIT)
        (IF USER::*DEBUG* (FORMAT T "~%   Entered function 'write_mission.orders'."))
        (UNITMSG   'mission.orders 'init-orders)
        (PUT.VALUE 'mission.orders 'active-mission    THISUNIT)
        (PUT.VALUE 'mission.orders 'action            (GET.VALUE THISUNIT 'action))
        (PUT.VALUE 'mission.orders 'area-operation    (GET.VALUE THISUNIT 'area-operation))
        (PUT.VALUE 'mission.orders 'class             (GET.VALUE THISUNIT 'class))
        (PUT.VALUE 'mission.orders 'goal-posn         (GET.VALUE THISUNIT 'goal-posn))
        (PUT.VALUE 'mission.orders 'hovering-mode     (GET.VALUE THISUNIT 'hovering-mode))
        (PUT.VALUE 'mission.orders 'initial-heading   (GET.VALUE THISUNIT 'initial-heading))
        (PUT.VALUE 'mission.orders 'mission-depth     (GET.VALUE THISUNIT 'mission-depth))
        (PUT.VALUE 'mission.orders 'mission-speed     (GET.VALUE THISUNIT 'mission-speed))
        (PUT.VALUE 'mission.orders 'safety-radius     (GET.VALUE THISUNIT 'safety-radius))
        (PUT.VALUE 'mission.orders 'start-posn        (GET.VALUE THISUNIT 'start-posn))
        (PUT.VALUE 'mission.orders 'threat            (GET.VALUE THISUNIT 'threat))
        (PUT.VALUE 'mission.orders 'time-available
                                    (* 60.0 (GET.VALUE THISUNIT 'time-available)))
        (IF USER::*DEBUG* (FORMAT T "~%   Exit function 'write_ops.orders'.")) )


;===============================================================================
; MISSION.ORDERS UNIT methods
;===============================================================================

  ;-----------------------------------------------------------------------------
  ;  INIT_ORDERS method is defined for [unit::slot]=[MISSION.ORDERS::init-orders].
  ;                                       -- called by write_mission.orders
  ;-----------------------------------------------------------------------------
  (DEFUN init_orders (THISUNIT)
        (IF USER::*DEBUG* (FORMAT T "~%   Entered function 'init_orders'."))
        (REMOVE.ALL.VALUES THISUNIT 'active-mission)
        (REMOVE.ALL.VALUES THISUNIT 'action)
        (REMOVE.ALL.VALUES THISUNIT 'area-operation)
        (REMOVE.ALL.VALUES THISUNIT 'class)
        (REMOVE.ALL.VALUES THISUNIT 'goal-posn)
        (REMOVE.ALL.VALUES THISUNIT 'hovering-mode)
        (REMOVE.ALL.VALUES THISUNIT 'initial-heading)
        (REMOVE.ALL.VALUES THISUNIT 'mission-depth)
        (REMOVE.ALL.VALUES THISUNIT 'mission-speed)
        (REMOVE.ALL.VALUES THISUNIT 'safety-radius)
        (REMOVE.ALL.VALUES THISUNIT 'start-posn)
        (REMOVE.ALL.VALUES THISUNIT 'threat)
        (REMOVE.ALL.VALUES THISUNIT 'time-available)
        (IF USER::*DEBUG* (FORMAT T "~%   Exit function 'init_orders'."))
        (init_auv_status_panel) )
  (DEFUN init_auv_status_panel ()
        (REMOVE.ALL.VALUES 'auv.status 'x-posn)
        (REMOVE.ALL.VALUES 'auv.status 'y-posn)
        (REMOVE.ALL.VALUES 'auv.status 'depth-under-sub)
        (REMOVE.ALL.VALUES 'auv.status 'depth)
        (REMOVE.ALL.VALUES 'auv.status 'heading)
        (REMOVE.ALL.VALUES 'auv.status 'rpm) )


(DEFUN compute_score (THISUNIT)
      (KEE::REMOVE.ALL.VALUES KEE::'decision.maker KEE::'best-score)
      (KEE::PUT.VALUE KEE::'decision.maker KEE::'best-score 0.0)
      (KEE::PUT.VALUE KEE::'decision.maker KEE::'astar-score
                        (+ (KEE::GET.VALUE KEE::'planner KEE::'astar-planning-time)
                           (KEE::GET.VALUE KEE::'planner KEE::'astar-space-constraint)
                           (KEE::GET.VALUE KEE::'planner KEE::'astar-path-optimality) ))
      (KEE::PUT.VALUE KEE::'decision.maker KEE::'bfirst-score
                        (+ (KEE::GET.VALUE KEE::'planner KEE::'bfirst-planning-time)
                           (KEE::GET.VALUE KEE::'planner KEE::'bfirst-space-constraint)
                           (KEE::GET.VALUE KEE::'planner KEE::'bfirst-path-optimality) ))
      (KEE::PUT.VALUE KEE::'decision.maker KEE::'hsearch-score
                        (+ (KEE::GET.VALUE KEE::'planner KEE::'hsearch-planning-time)
                           (KEE::GET.VALUE KEE::'planner KEE::'hsearch-space-constraint)
                           (KEE::GET.VALUE KEE::'planner KEE::'hsearch-path-optimality) )) )
```

```
;;; -*- Package: KEE; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;****************************************************************************
;
;  Filename.......:   mission-agents.lisp
;  Author........ :   Ong Seow Meng
;
;  Date Created...:   20 Jan 90
;  Description.....:  Contains the methods referenced by the following UNITs in
;                     the MPES knowledge base.
;                     [UNIT]=[PLANNER]
;                     [UNIT]=[CONSTRUCTOR]
;                     [UNIT]=[EXECUTOR]
;
;  Modifications.:
;
;****************************************************************************


;============================================================================
; PLANNER methods
;============================================================================

   ;-------------------------------------------------------------------------
   ;  PLAN method is [PLANNER::plan]
   ;-------------------------------------------------------------------------
   (DEFUN plan (THISUNIT)
          (IF USER::*DEBUG* (FORMAT T "~%   Entered function 'plan'."))
          (RETRACT '(TEXT 'planning-phase))
          (IF USER::*DEBUG* (FORMAT T "~%   before rule base activiation"))
          ;; activate mission.planning.rules which operates on slots in ops.orders unit.
          (UNITMSG 'knowledge.processor 'start)
          (UNITMSG 'voters 'start)
          (UNITMSG 'decision.maker 'start)
          (UNITMSG THISUNIT 'generate-construction-orders)
        ;; (ASSERT NIL 'mission.planning.rules NIL :AGENDA.CONTROLLER 'GREATEST.WEIGHT)
          (IF USER::*DEBUG* (FORMAT T "~%   after rule base activation.")))


  ;-------------------------------------------------------------------------
  ; generate_construction_orders   --  for [slot]=[generate-construction-orders]
  ;-------------------------------------------------------------------------
  (DEFUN generate_construction_orders (THISUNIT)
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'active-mission
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'active-mission) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'area-operation
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'area-operation) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'goal-posn
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'goal-posn) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'hovering-mode
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'hovering-mode) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'mission-depth
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-depth) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'mission-speed
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-speed) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'safety-radius
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'safety-radius) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'start-posn
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'start-posn) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'threat
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'threat) )
         (KEE::PUT.VALUE KEE::'construction.orders KEE::'path-plan-method
                         (KEE::GET.VALUE KEE::'planner KEE::'recommended-path-planner) ) )
```

```
;------------------------------------------------------------------------------
; start_knowledge.processor  --  for [UNIT::slot]=[KNOWLEDGE.PROCESSOR::start]
;------------------------------------------------------------------------------
(DEFUN start_knowledge.processor (THISUNIT)
       (ASSERT NIL THISUNIT) )

;------------------------------------------------------------------------------
; start_decision.maker  --  for [UNIT::slot]=[DECISION.MAKER::start]
;------------------------------------------------------------------------------
(DEFUN start_decision.maker (THISUNIT)
       (UNITMSG THISUNIT 'compute-score)
       (ASSERT NIL THISUNIT) )

;------------------------------------------------------------------------------
; start_voters  --  for [UNIT::slot]=[VOTERS::start]
;------------------------------------------------------------------------------
(DEFUN start_voters (THISUNIT)
       (ASSERT NIL THISUNIT) )


;================================================================================
; CONSTRUCTOR methods
;================================================================================

  ;------------------------------------------------------------------------------
  ;  CONSTRUCT method is [CONSTRUCTOR::construct]
  ;------------------------------------------------------------------------------
  (DEFUN construct (THISUNIT)
         (IF USER::*DEBUG* (FORMAT T "~%   Entered function 'construct'."))
         (RETRACT '(TEXT 'construction-phase))
       ;; (FORMAT T "~%CONSTRUCTION phase in progress.....")

         (UNITMSG (GET.VALUE 'mission.orders 'active-mission) 'construct-mission))

     ;;    (ASSERT '(TEXT 'execution-phase) 'mission.controller) )


;================================================================================
; EXECUTOR methods
;================================================================================

  ;------------------------------------------------------------------------------
  ;  EXECUTE method is [EXECUTOR::execute]
  ;------------------------------------------------------------------------------
  (DEFUN execute (THISUNIT)
         (IF USER::*DEBUG* (FORMAT T "~%   Entered function 'execute'."))
         (RETRACT '(TEXT 'execution-phase))
         (UNITMSG (GET.VALUE 'mission.orders 'active-mission) 'execute-mission)
       ;; (FORMAT T "~%MISSION EXECUTION in progress......")
         (IF USER::*DEBUG* (FORMAT T "~%   end of execute function")) )

  ;------------------------------------------------------------------------------
  ;  ABORT method is [EXECUTOR::abort-mission]
  ;------------------------------------------------------------------------------
  (DEFUN abort_mission (THISUNIT)
         (clear.unstructured.facts)
         (UNITMSG 'viewport-execute.abort.panel.16 'close-panel!)
         (UNITMSG 'panels 'reset-screen)
         (init_auv_status_panel)
         (IF USER::*iris-sym-comms-established* (USER::end-con)) )
```

```
;;; -*- Package: USER; Mode: LISP; Syntax: Common-Lisp; Base: 10 -*-

;*******************************************************************************
;
;  Filename......:  umissions.lisp
;  Author........:  Ong Seow Meng
;
;  Date Created..:  24 Jan 90
;  Description...:  Contains the methods referenced by
;                   [UNIT::SLOT]=[<specific-mission-unit>::construct-mission]
;                   [UNIT::SLOT]=[<specific-mission-unit>::execute-mission].
;
;  Modifications.:
;
;
;*******************************************************************************


;===============================================================================
; TRANSIT_POOL UNIT methods
;===============================================================================

  ;-----------------------------------------------------------------------------
  ; INIT_USER_PKG method is defined for [UNIT::SLOT]=[TRANSIT.POOL::init-user-pkg].
  ;      It is called by the construct_transit_pool method in [UNIT]=[TRANSIT.POOL].
  ;-----------------------------------------------------------------------------
  (DEFUN init_user_pkg (THISUNIT)
        (IF *DEBUG* (FORMAT T "~%   Entered function 'init_user_pkg'."))
        (init-display)
        (init_global_variables)
        (SETQ *pooldepth* (* *zmapsize* *real-vert-dist-pu-coord*))
        (SETF *goal* (change_to_path_planning_coord
                         (KEE::GET.VALUE KEE::'mission.orders KEE::'goal-posn) ) )
        (FORMAT T "~%    *goal* =  ") (PRINC *goal*)
        ;;(LET ( (initial-hdg (* *deg-to-rad-factor*
        ;;                         (KEE::GET.VALUE KEE::'mission.orders KEE::'initial-heading) )))
        ;;      (SETF *start*
        ;;            (CONS initial-hdg (LIST (change_to_path_planning_coord
        ;;                     (KEE::GET.VALUE KEE::'mission.orders KEE::'start-posn) ))) ) )
        (SETF *start*
              (CONS 0 (LIST (change_to_path_planning_coord
                             (KEE::GET.VALUE KEE::'mission.orders KEE::'start-posn) ))) )
        (FORMAT T "~%    *start* =  ") (PRINC *start*)
        (SETF *mission-depth* (nearest_vert_coord
                              (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-depth)) )
        (FORMAT T "~%    *mission-depth* =  ") (PRINC *mission-depth*)
        (SETF *safety-dist* (nearest_horiz_coord
                              (KEE::GET.VALUE KEE::'mission.orders KEE::'safety-radius)) )
        (FORMAT T "~%    *safety-dist* =  ") (PRINC *safety-dist*)
        (TERPRI)
        (FORMAT T "~%Initialising system. Wait......... ")

        (make_emap)
        (init_pool_emap)
        (place_obs_ls)

        (SETF *goal-vicinity-list* (make_vicinity_list *goal*))
        (IF *DEBUG* (FORMAT T "~%   Exit function 'init_user_pkg'.")) )
```

```
;-----------------------------------------------------------------------------------
; INIT_GLOBAL_VARIABLES function is called by function init_user_pkg.
;-----------------------------------------------------------------------------------
(DEFUN init_global_variables ()
       (IF *DEBUG* (FORMAT T "~%  Entered function 'init_global_variables'."))
       (LET* ( (curr-area-ops    (KEE::GET.VALUE KEE::'mission.orders KEE::'area-operation))
               (curr-mission     (KEE::GET.VALUE KEE::'mission.orders KEE::'active-mission))
               (threat-level     (KEE::GET.VALUE curr-mission KEE::'threat)) )
             (SETF *xmapsize*    (KEE::GET.VALUE curr-area-ops KEE::'xmapsize))
             (SETF *ymapsize*    (KEE::GET.VALUE curr-area-ops KEE::'ymapsize))
             (SETF *zmapsize*    (KEE::GET.VALUE curr-area-ops KEE::'zmapsize))
             (SETF *ObstacleLs*  (KEE::GET.VALUE curr-area-ops KEE::'selected-obst))
             (SETF *Bottom-Search-Preferred* (IF (EQUAL threat-level KEE::'HOSTILE) T NIL)) )

       (IF *DEBUG* (PROGN (FORMAT T "~%   *Bottom-Search-Preferred* = ")
                          (PRINC *Bottom-Search-Preferred*) ))
       (SETF *path*               NIL)
       (SETF *real-path*          NIL)
       (SETF *return-path*        NIL)
       (SETF *goal-vicinity-list* NIL)
       (SETF *Obstacle-Mode*      NIL)
       (SETF *Near-Obst-Edge*     NIL)
       (SETF *search-mode* 'fwd-level)
       (SETF *Current-Mode* 'Normal-Mode)
       (SETF *curr-speed*         0.0)
       (SETF *iris-sym-comms-established* NIL) )


;-----------------------------------------------------------------------------------
; CONSTRUCT_TRANSIT_POOL method is for [unit::slot]=[TRANSIT.POOL::construct-mission].
;-----------------------------------------------------------------------------------
(DEFUN construct_transit_pool (THISUNIT)
       (IF *DEBUG* (FORMAT T "~%  Entered construct_transit_pool function."))
       (KEE::UNITMSG THISUNIT KEE::'init-user-pkg)
       (TERPRI)
    ;; (FORMAT T "Change path-plan-method in unit mission.plan now (if required).") (READ)
      (LET* ((searchmethod   (KEE::GET.VALUE KEE::'construction.orders KEE::'path-plan-method))
             (searchfunction (KEE::GET.VALUE searchmethod KEE::'user-function))
             ; goal-posn is in real coord !!!!
             (goal-posn      (KEE::GET.VALUE KEE::'mission.orders KEE::'goal-posn))
             (transit-depth  (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-depth))
             (transit-speed  (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-speed)) )

         (IF *DEBUG*
             (PROGN  (FORMAT T "~%   searchmethod is ")  (PRINC searchmethod)  (TERPRI)
           (FORMAT T "~%   goal-posn is ")      (PRINC goal-posn)      (TERPRI)
                     (FORMAT T "~%   transit-depth is ") (PRINC transit-depth) (TERPRI)
                     (FORMAT T "~%   transit-speed is ") (PRINC transit-speed) (TERPRI) ) )

         ; Bind global variables
         (LET ( (start-real-coord (real_posn_coord (posn *start*))) )
              (SETQ xstart (x_coord start-real-coord))
              (SETQ x xstart)
              (SETQ ystart (y_coord start-real-coord))
              (SETQ y ystart)
              (SETQ zstart (z_coord start-real-coord))
              (SETQ z zstart)
              (SETQ depth_under_sub (- *pooldepth* (z_coord start-real-coord))) )
        (SETQ sub_depth (* *real-vert-dist-pu-coord* (z_coord (posn *start*))))

        (FORMAT T "~%  Path planning begins......")
        ; Begin Path Planning
        (SETF *max-qlength* 1)
        (TIME (SETQ *real-path* (plan_path searchfunction *start*)))
        (KEE::PUT.VALUE KEE::'mission.details KEE::'path *real-path*)
        (SETQ *return-path* (REVERSE *real-path*))
        (print_performance_data)
```

```
                ; Display start, goal and path on color monitor.
                ; NOTE: The monitor coord system is opposite that of iris (x-iris = y-monitor)
                (draw-start-pos ystart xstart zstart)
                (draw-goal-pos (y_coord goal-posn) (x_coord goal-posn) (z_coord goal-posn))
                (move-icon ystart xstart zstart)
                (display_path_on_monitor)
                (TERPRI)
                (PRINC "Detailed MISSION PLAN ready for execution.")
                (TERPRI)
                (IF *DEBUG* (FORMAT T "~%   Exit construct_transit_pool function.")) ) )


(DEFUN print_performance_data ()
       (TERPRI)
       (PRINC " MAX QUEUE length = ") (PRINC *max-qlength*) (TERPRI)
       (FORMAT T "~% Cost of Path      = ") (PRINC (cost_of_path *path*)) (TERPRI) )


(DEFUN send_search_parameters_to_IRIS ()
       ; Initiate conversation with IRIS
       (SETF talk (make-instance 'conversation-with-iris))
       (choose-iris 'iris5)
       (start-con)
       (TERPRI)
       (SETF *iris-sym-comms-established* T)
       (PRINC "Connection with iris established.")
       (TERPRI)

       ; send obstacles to iris
       (LET ( (obst-posn-list (REST (FIRST *ObstacleLs*))) )
            (send_float (LENGTH obst-posn-list))
            (PRINT obst-posn-list)
            (MAPCAR #'send_obstacles_to_iris
                    (LIST (CONS (FIRST (FIRST *ObstacleLs*))
                                (MAPCAR #'real_posn_coord obst-posn-list)))) )

       ; Send initial state to IRIS
       (send_float xstart)
       (FORMAT T "~% xstart sent to iris:  ") (PRINC xstart)
       (TERPRI)
       (send_float ystart)
       (FORMAT T "~% ystart sent to iris:  ") (PRINC ystart)
       (TERPRI)
       (send_float zstart)
       (FORMAT T "~% zstart sent to iris: ") (PRINC zstart)
       (TERPRI)
       (LET ((init-dir (* *rad-to-deg-factor* (direction *start*))))
            (send_float init-dir)
            (FORMAT T "~% initial direction sent to iris: ")
            (PRINC init-dir)
            (TERPRI) )
       (FORMAT T "~% Initial AUV State sent to iris.")

       ; Send path to IRIS
       (send_float (LENGTH *real-path*))
       (MAPCAR #'send_state_to_iris *real-path*)

       ; Send goal location to IRIS.
       (LET ( (goal-posn (KEE::GET.VALUE KEE::'mission.orders KEE::'goal-posn)) )
            (send_float (x_coord goal-posn))
            (send_float (y_coord goal-posn)) ) )


 (DEFUN plan_path (searchmethod start-state)
        (SETF *goal-vicinity-list* (make_vicinity_list *goal*))
        (MAPCAR #'change_to_real_state_coord
                (process_path (APPEND (funcall searchmethod) (LIST (LIST 0 *goal*)))) ) )
```

```
(DEFUN process_path (path)
       (COND ( (NULL (CDR path)) path )
             ( T  (LET ( (curr-state (FIRST path))
                         (next-state (SECOND path)) )
                      (IF (sharp_turn curr-state next-state)
                          (process_path (REST path))
                          (CONS curr-state (process_path (REST path))) ) )) ) )

(DEFUN sharp_turn (curr-state next-state)
       (COND ( (course_change_90_degrees curr-state next-state)
                      (IF (OR (x_coord_unchanged curr-state next-state)
                              (y_coord_unchanged curr-state next-state) )
                          T
                          NIL ) )
             ( T NIL ) ) )

(DEFUN course_change_90_degrees (curr-state next-state)
       (LET ( (curr-hdg (FIRST curr-state))
              (next-hdg (FIRST next-state)) )
            (IF (>= (ABS (- next-hdg curr-hdg)) *half-PI*)
                T
                NIL ) ) )
```

```
(DEFUN x_coord_unchanged (curr-state next-state)
       (IF (= 0 (xcoord_diff (posn curr-state) (posn next-state)))
           T
           NIL ) )

(DEFUN y_coord_unchanged (curr-state next-state)
       (IF (= 0 (ycoord_diff (posn curr-state) (posn next-state)))
           T
           NIL ) )

;-----------------------------------------------------------------------------------
; EXECUTE_TRANSIT_POOL method is defined for [unit::slot]=(transit_pool::execute-mission].
;-----------------------------------------------------------------------------------
(DEFUN execute_transit_pool (THISUNIT)
       (KEE::UNITMSG KEE::'viewport-auv.status.panel.2 KEE::'open-panel!)
       (IF *DEBUG* (FORMAT T "~%   Entered function 'execute_transit_pool'."))
       (send_search_parameters_to_IRIS)
       (TERPRI) (PRINC "Hit a key on Iris5 main terminal to continue.") (TERPRI)
       (LET ( (transit-speed (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-speed)) )
            (transit_without_contacts (real_posn_coord *goal*) transit-speed "TRANSIT")
            (transit_back_without_contacts transit-speed) )
            (TERPRI)
            (PRINC "TRANSIT_POOL MISSION COMPLETED.")
            (TERPRI)
            (stop_in_pool xstart ystart)
            (end-con)
       (IF *DEBUG* (FORMAT T "~%   Exit function 'execute_transit_pool'.")) )
```

```
(DEFUN transit_without_contacts (goal-posn transit-speed sub-command)
       ;; goal-posn is in real distance coordinates
       ;; *real-path* is in real distance coordinates
       (DO* ( (curr-posn (LIST x y sub_depth) (LIST x y sub_depth))
              (horiz-dist (horiz_coord_dist curr-posn goal-posn)
                          (horiz_coord_dist curr-posn goal-posn) )
              (vert-dist  (abs_vert_coord_dist curr-posn goal-posn)
                          (abs_vert_coord_dist curr-posn goal-posn) ) )
            ( (AND (< horiz-dist *real-horiz-dist-pu-coord*)
                   (< vert-dist  *real-vert-dist-pu-coord*) )
              (TERPRI) (PRINC "AUV AT GOAL") (TERPRI) )

            (LET* ( (next-subgoal (posn (SECOND *real-path*)))
                    (xsubgoal (x_coord next-subgoal))
                    (ysubgoal (y_coord next-subgoal))
                    (zsubgoal (z_coord next-subgoal))
                    (newspeed *curr-speed*) )
                  ;;; (zsubgoal (- *pooldepth* (z_coord next-subgoal))) )
                 (SETF newspeed (adjust_speed transit-speed))
                 (SETF *curr-speed* newspeed)
                 (SETQ autocourse (get_autocourse x y xsubgoal ysubgoal))
                 (send_float autocourse)
                 (send_float zsubgoal)
                 (send_float newspeed)
                 (send_float xsubgoal)
                 (send_float ysubgoal)
                 (send_string sub-command)
                 (TERPRI)
                 (get_data_from_iris_without_contacts)
                 (COND ( (AND (> *real-horiz-dist-pu-coord*
                                 (horiz_coord_dist curr-posn next-subgoal) )
                              (> *real-vert-dist-pu-coord*
                                 (abs_vert_coord_dist curr-posn next-subgoal) ) )
                         (SETQ *real-path* (REST *real-path*)) ) ) ) ) )


(DEFUN transit_back_without_contacts (transit-speed)
       (SETQ *real-path* *return-path*)
       (transit_without_contacts (real_posn_coord (posn *start*))
                                 transit-speed
                                 "TRANSIT BACK") )


(DEFUN stop_in_pool (xstart ystart)
       (FORMAT T "~%   Standing by for Recovery......")
       (DO ((numtimes 1 (1+ numtimes)))
           ((= numtimes 50))
           (send_float (get_autocourse x y xstart ystart))
           (send_float 0)                                    ; put auv on surface.
           (send_float 0)                                    ; come to all stop.
           (send_float xstart)
           (send_float ystart)
           (send_string "STANDING BY FOR RECOVERY.")
           (get_data_from_iris_without_contacts) ) )
```

```
  (DEFUN get_data_from_iris_without_contacts()
         (SETQ x (get_data))
         (KEE::PUT.VALUE KEE::'auv.status KEE::'x-posn x)
         (SETQ y (get_data))
         (KEE::PUT.VALUE KEE::'auv.status KEE::'y-posn y)
         (SETQ depth_under_sub (get_data))
         (KEE::PUT.VALUE KEE::'auv.status KEE::'depth-under-sub depth_under_sub)
         (SETQ sub_depth (get_data))
         (KEE::PUT.VALUE KEE::'auv.status KEE::'depth sub_depth)
         (SETQ acourse (get_data))
         (KEE::PUT.VALUE KEE::'auv.status KEE::'heading acourse)
         (KEE::PUT.VALUE KEE::'auv.status KEE::'rpm
                           (KEE::GET.VALUE KEE::'mission.orders KEE::'mission-speed))
         (PRINC "   x          y     depth_under_auv  auv's depth  course")
         (FORMAT T "~% ~0,2F ~10,2F ~12,2f ~12,2F ~12,2F" x y depth_under_sub sub_depth acourse)
         ;;  The following few line transfer data to the color monitor
         ;;  color monitor coord system is opposite that of iris display (x-iris = y-monitor)
         (move-icon y x sub_depth)
         (TERPRI))

;*************************************************************************************
;
; Functions to support transit_pool mission.
;
;*************************************************************************************

(DEFUN adjust_speed (transit-speed)
       (COND ((< (LENGTH *real-path*) 3) (MIN 250.0 transit-speed))
             ((moving_vertically_soon) *vert-mvt-speed*)
             ((turning_vertically_soon) (MIN *vert-turning-speed* transit-speed))
             (T (MIN transit-speed (+ (* 0.3 *curr-speed*) (* 0.7 transit-speed))) ) ) )

(DEFUN moving_vertically_soon ()
       (LET ((next-posn (posn (SECOND *real-path*)))
             (next2posn (posn (THIRD  *real-path*))) )
            (IF (< (horiz_coord_dist next-posn next2posn) 0.2)
                T
                NIL) ) )

(DEFUN turning_vertically_soon ()
       (LET ((next-posn (posn (SECOND *real-path*)))
             (next2posn (posn (THIRD  *real-path*))) )
            (IF (AND (>= (horiz_coord_dist next-posn next2posn)
                         *real-horiz-dist-pu-coord*)
                     (> (abs_vert_coord_dist next-posn next2posn) 0.2) )
                T
                NIL) ) )

(DEFUN change_to_path_planning_coord (real-posn-coord)
       (LIST (nearest_horiz_coord (x_coord real-posn-coord))
             (nearest_horiz_coord (y_coord real-posn-coord))
             (nearest_vert_coord  (z_coord real-posn-coord)) ) )


(DEFUN nearest_horiz_coord (real-dist)
       (MAX 1 (round (/ (- real-dist *approx-half-real-horiz-dist-pu-coord*)
                        *real-horiz-dist-pu-coord*) )) )


(DEFUN nearest_vert_coord (real-height)
       (MAX 1 (round (/ real-height *real-vert-dist-pu-coord*))) )


(DEFUN change_to_real_state_coord (state)
       (CONS (direction state) (LIST (real_posn_coord (posn state)))) )
```

```
(DEFUN real_posn_coord (posn-coord)
       (LIST (real_horiz_dist (x_coord posn-coord))
             (real_horiz_dist (y_coord posn-coord))
            ;; the following needs to be changed later to get rid of *pooldepth*
            ;;  (- *pooldepth* (real_vert_dist  (z_coord posn-coord))) ) )
              (real_vert_dist (z_coord posn-coord)) ) )


(DEFUN real_horiz_dist (coord-value)
       (* coord-value *real-horiz-dist-pu-coord*) )


(DEFUN real_vert_dist (coord-value)
       (* coord-value *real-vert-dist-pu-coord*) )


(DEFUN get_autocourse (x y xl yl)
   (cond
     ((< x xl) (autocoursel x y xl yl))
     (t (- 360 (autocoursel x y xl yl)))))


(DEFUN autocoursel (x y xl yl)
   (* 57.295 (acos (/ ( - yl y)
                      (get_the_distance x y xl yl)))))


(DEFUN get_the_distance (x y xl yl)
   (sqrt (+ (sqr (- x xl))
            (sqr (- y yl)))))


(DEFUN display_path_on_monitor  ()
       (MAPCAR #'plot_point *real-path*) )


(DEFUN plot_point (state)
       (draw-path-pos (y_coord (posn state))
                      (x_coord (posn state))
                      (z_coord (posn state)) ) )
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5100	2
3.	Chief Of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, DC 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Curricular Officer, Code 37 Computer Technology Program Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Professor Robert B. McGhee Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	24

7.	Professor Se-Hung Kwak, Code 52Kw Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	5
8.	Professor Neil Rowe Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
9.	Professor A. J. Healey, Code 69Hy Mechanical Engineering Department Naval Postgraduate School Monterey, CA 93943-5000	1
10.	Professor R. Christi, Code 62Cx Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943-5000	1
11.	United States Military Academy Department of Geography & Computer Science ATTN: CPT Mark Fichten West Point, NY 10996-1695	1
12.	Naval Ocean Systems Center Ocean Engineering Division (Code 94) ATTN: Paul Heckman San Diego, CA 92152-5000	1
13.	Naval Coastal System Center Navigation, Guidance, and Control Branch ATTN: G. Dobeck Panama City, FL 32407-5000	1

14.	Naval Surface Warfare Center ATTN: Hal Cook, Code u25 White Oak, MD 20910	1
15.	HQDA Artificial Intelligence Center ATTN: DACS-DMA, LTC A. Anconetoni The Pentagon, Room 1D659 Washington, D.C. 20310-0200	1
16.	RADM G. Curtis, Code PMS-350 Naval Sea Systems Command Washington, D.C. 20362-5101	1
17.	Mr. Ong Seow Meng #09-102, Block 272 Yishun St. 22 Singapore 2776 Republic of Singapore	1
18.	Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943-5000	1
19.	NASA Goddard Space Flight Center ATTN: Russell Werneth Greenbelt Road Greenbelt, MD 20771	1
20.	MARINTEK ATTN: Svein Kristiansen Haakon Haakonsons gt. 34 P.O. Box 4125 Valentinlyst N-7000 Trondheim, Norway	1